Programmer's Reference

Publication Number 54710-97002
Fifth Edition, October 1993

This reference applies directly to firmware revision code 3.XX.

For Safety information, Warranties, and Regulatory information, see the pages behind the index.

HP 54710 and HP 54720 Oscilloscopes

The HP 54710 and HP 54720 Oscilloscope Programming Command Set

:(root)

Common Commands

- *CLS
- *ESE
- *ESR?
- *IDN?
- *LRN?
- *OPC?
- *OPT?
- *RCL
- *RST
- *SAV
- *SRE
- *STB?
- *TRG
- *TST?
- *WAI

AER?
AUToscale
BLANk
CDISplay
DIGitize
ERASe
HEEN
HER?
LER?
LTEE
LTER?
MENU
MERGe
MODel?
MTEE
MTER?
OPEE
OPER?
PRINt
RECall
RUN
SERial
SINGle
STOP
STORe
TER?
UEE
UER?
VIEW

SYStem:
DATE
DSP
ERRor?
HEADer
KEY
LONGform
SETup
TIME

ACQuire:
BWlimit
COMPlete
COUNt
INTerpolate
MODE
POINts
SRATe
TYPE

CALibrate:
BEST
FRAMe
OUTPut
PLUGin
PROBe
SKEW
STATus

CHANnel:
BWLimit
DISPlay
INPut
OFFSet
OUTPut
PROBe
PROTection
RANGe
SCALe
SENSitivity
UNITs

DISK:
DELete
DIRectory?
FORMat
LOAD
STORe

DISPlay:
ASSign
CGRade
COLumn
DATA
DCOLor
DWAVeform
FORMat
GRATicule
INVerse
LINE
MASK
PERSistence
ROW
SCOLor
SOURce
STRing
TEXT

FFT
DISPlay
FREQuency
MAGNify
MSPan
OFFSet
RANGe
RESolution
SOURce
SPAN
WINDow

FUNCtion<N>
ADD
DIFF
DISPlay
DIVide
FFT
FFTMagnitude
HORizontal
INTegrate
INVert
MAGNify
MAXimum
MINimum
MULTiply
OFFSet
ONLY
RANGe
SUBTract
VERSus
VERTical

HARDcopy:
ADDress
AREA
BACKground
DESTination
DEVice
FACTors
FFEed
FILEname
LENGth
MEDia

HISTogram:
AXIS
MODE
RUNTil
SCALe
WINDow

LTESt:
FAIL
LLIMit
MNFound
RUN
SOURce
SSCreen
SSUMmary
SWAVeform
TEST
ULIMit

Marker:
CURSor?
MEASurement
MODE
TDELta?
TSTArt
TSTOp
VDELta?
VSTArt
VSTOp
X1Position
X2Position
X1Y1source
X2Y2source
XDELta?
Y1Position
Y2Position
YDELta?

54710B03

MEASure:	MTESt:	Timebase:	TRIGger:	TRIGger<N>:	PMEMory:	WMEMory<N>:	WAVeform:
DEFine	AMASk	DELay	DEVents	BWLimit	ADD	DISPlay	BANDpass?
DELTatime	COUNt	POSition	DTIMe	PROBe	CLEAR	SAVe	BYTeorder
DUTycycle	MASK	RANGe	EDGe		DISPlay	XOFFset	COMPlete?
FALLtime	POLYgon	REFerence	GLITch		ERASe	XRANge	COUNt?
FFT	RUMode	SCALe	HOLDoff		MERGe	YOFFset	COUPling?
FREQuency	SCALe	VIEW	HYSTeresis			YRANge	DATA
HISTogram	SSCReen	WINDow	LEVel				FORMat
NWIDth	SSUMmary		MODE				POINts?
OVERshoot	SWAVeform		PATTern				PREamble
PERiod	TEST		SLOPe				SOURce
PREShoot			SOURce				TYPE?
PWIDth			STATe				VIEW
RESults?			SWEep				XDISplay?
RISetime			STV				XINCrement?
SCRatch			SWEep				XORigin?
SENDvalid			UDTV				XRANge?
SOURce							XREFerence?
STATistics							XUNits?
TEDGe							YDISplay?
TMAX							YINCrement?
TMIN							YORigin?
TVOLt							YRANge?
VAMPlitude							YREFerence?
VAVerage							YUNits?
VBASe							
VLOWer							
VMAX							
VMIDdle							
VMIN							
VPP							
VRMS							
VTIMe							
VTOP							
VUPper							

54710804

In This Book

This book is your guide to programming the HP 54710 and HP 54720 Digitizing Oscilloscopes using the HP-IB command set.

Part One, "Introduction to Programming the HP 54710/HP 54720 Oscilloscopes," gives you the conceptual information needed to start programming the oscilloscope. This part includes information about basic program communications, interface, syntax, data types, and status reporting. It also has a set of sample programs that show you some typical applications.

Part Two, "HP 54710/HP 54720 HP-IB Command Reference," describes all the commands used to program the oscilloscope. Each chapter lists the commands that belong to an individual subsystem, and explains the function of each command.

Contents

Part 1 Introduction to Programming the HP 54710/HP 54720 Oscilloscopes

1 Introduction to Programming

5 Programming Syntax

6 Programming Conventions

7 Example Programs

Part 2 HP 54710/HP 54720 HP-IB Command Reference

17 Hardcopy Commands

18 Marker Commands

19 Measure Commands

Part 1

Introduction to Programming the HP 54710/HP 54720 Oscilloscopes

1

Introduction to Programming

Introduction to Programming

This chapter introduces the basics for remote programming of an oscilloscope. The programming instructions in this manual conform to the IEEE 488.2 Standard Digital Interface for Programmable Instrumentation. The programming instructions provide the means of remote control.

There are basic operations that can be done with a controller and an oscilloscope:

- Set up the instrument.
- Make measurements.
- Get data (waveform, measurements, configuration) from oscilloscope.
- Send information (pixel image, configurations) to oscilloscope.

Other tasks are accomplished by combining these basic functions.

> The programming examples for individual commands in this manual are written in HP BASIC 5.0 for an HP 9000 Series 200/300 Controller.

Talking to the Instrument

Computers acting as controllers communicate with the instrument by sending and receiving messages over a remote interface. Instructions for programming normally appear as ASCII character strings embedded inside the output statements of a "host" language available on your controller. The input statements of the host language are used to read in responses from the oscilloscope.

For example, HP 9000 Series 200/300 BASIC uses the OUTPUT statement for sending commands and queries. After a query is sent, the response is usually read in using the ENTER statement.

Messages are placed on the bus using an output command and passing the device address, program message, and terminator. Passing the device address ensures that the program message is sent to the correct interface and instrument.

The following HP BASIC OUTPUT statement sends a command that sets the bandwidth limit of channel 1 to on:

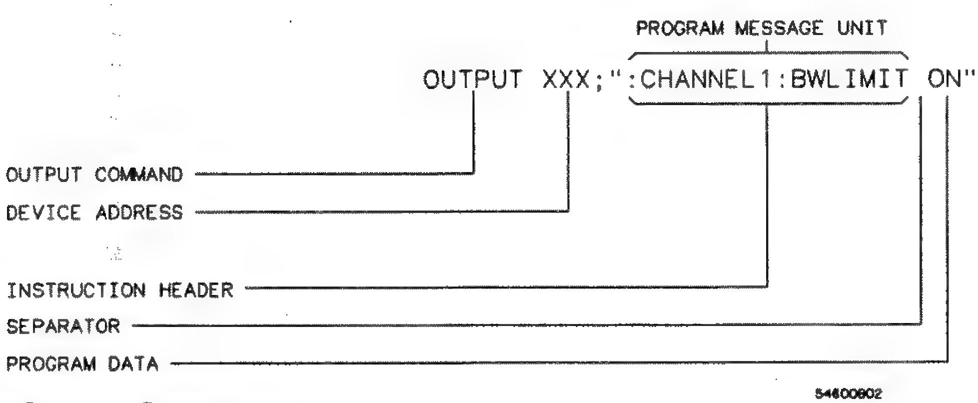
```
OUTPUT < device address > ;":CHANNEL1:BWLIMIT ON"<terminator>
```

The < device address > represents the address of the device being programmed. Each of the other parts of the above statement are explained in the following pages.

Program Syntax

To program the instrument remotely, you must have an understanding of the command format and structure expected by the instrument. The IEEE 488.2 syntax rules govern how individual elements such as headers, separators, program data, and terminators may be grouped together to form complete instructions. Syntax definitions are also given to show how query responses are formatted. Figure 1-1 shows the main syntactical parts of a typical program statement.

Figure 1-1



Program Statement Syntax

Output Command

The output command is entirely dependent on the programming language. Throughout this manual HP 9000 Series 200/300 BASIC 5.0 is used in the examples of individual commands. If you are using other languages, you will need to find the equivalents of HP BASIC commands like OUTPUT, ENTER, and CLEAR in order to convert the examples. The instructions listed in this manual are always shown between quotes in the example programs.

Device Address

The location where the device address must be specified is also dependent on the programming language you are using. In some languages, this may be specified outside the output command. In HP BASIC, this is always specified after the keyword OUTPUT. The examples in this manual assume the oscilloscope is at device address 707. When writing programs, the address varies according to how the bus is configured.

Instructions

Instructions (both commands and queries) normally appear as a string embedded in a statement of your host language, such as BASIC, Pascal, or C. The only time a parameter is not meant to be expressed as a string is when the instruction's syntax definition specifies <block data>, such as learnstring. There are only a few instructions that use block data.

Instructions are composed of two main parts:

- The header, which specifies the command or query to be sent.
- The program data, which provide additional information needed to clarify the meaning of the instruction.

Instruction Header

The instruction header is one or more mnemonics separated by colons (:) that represent the operation to be performed by the instrument. The command tree in figure 5-1 illustrates how all the mnemonics can be joined together to form a complete header (see the "Programming Conventions" chapter).

The example in figure 1-1 is a command. Queries are indicated by adding a question mark (?) to the end of the header. Many instructions can be used as either commands or queries, depending on whether or not you have included the question mark. The command and query forms of an instruction usually have different program data. Many queries do not use any program data.

White Space (Separator)

White space is used to separate the instruction header from the program data. If the instruction does not require any program data parameters, you do not need to include any white space. In this manual, white space is defined as one or more spaces. ASCII defines a space to be character 32 (in decimal).

Program Data

Program data are used to clarify the meaning of the command or query. They provide necessary information, such as whether a function should be on or off, or which waveform is to be displayed. Each instruction's syntax definition shows the program data, as well as the values they accept. The section "Program Data Syntax Rules" in this chapter has all of the general rules about acceptable values.

When there is more than one data parameter, they are separated by commas (,). Spaces can be added around the commas to improve readability.

Header Types

There are three types of headers:

- Simple Command headers.
- Compound Command headers
- Common Command headers.

Simple Command Header

Simple command headers contain a single mnemonic. AUTOSCALE and DIGITIZE are examples of simple command headers typically used in this instrument. The syntax is:

<program mnemonic><terminator>

When program data must be included with the simple command header (for example, :DIGITIZE CHAN1), white space is added to separate the data from the header. The syntax is:

<program mnemonic><separator><program data><terminator>

Compound Command Header

Compound command headers are a combination of two program mnemonics. The first mnemonic selects the subsystem, and the second mnemonic selects the function within that subsystem. The mnemonics within the compound message are separated by colons. For example:

To execute a single function within a subsystem:

:<subsystem>:<function><separator><program data><terminator>

(For example :CHANNEL1:BWLIMIT ON)

Combining Commands in the Same Subsystem

To execute more than one function within the same subsystem a semi-colon (;) is used to separate the functions:

:<subsystem>:<function><separator><data>;<function><separator><data>
<terminator>

(For example :CHANNEL1:COUPLING DC;BWLIMIT ON)

Common Command Header

Common command headers control IEEE 488.2 functions within the instrument (such as clear status). Their syntax is:

*<command header><terminator>

No space or separator is allowed between the asterisk (*) and the command header. *CLS is an example of a common command header.

Duplicate Mnemonics

Identical function mnemonics can be used for more than one subsystem. For example, the function mnemonic RANGE may be used to change the vertical range or to change the horizontal range:

:CHANNEL1:RANGE .4

sets the vertical range of channel 1 to 0.4 volts full scale.

:TIMEBASE:RANGE 1

sets the horizontal time base to 1 second full scale.

CHANNEL1 and TIMEBASE are subsystem selectors and determine which range is being modified.

Query Headers

Command headers immediately followed by a question mark (?) are queries. After receiving a query, the instrument interrogates the requested function and places the answer in its output queue. The answer remains in the output queue until it is read or another command is issued. When read, the answer is transmitted across the bus to the designated listener (typically a controller). For example, the query :

```
TIMEBASE:RANGE?
```

places the current time base setting in the output queue. In HP BASIC, the controller input statement:

```
ENTER < device address > ;Range
```

passes the value across the bus to the controller and places it in the variable Range.

Query commands are used to find out how the instrument is currently configured. They are also used to get results of measurements made by the instrument. For example, the command:

```
:MEASURE:RISETIME?
```

instructs the instrument to measure the rise time of your waveform and place the result in the output queue.

The output queue must be read before the next program message is sent. For example, when you send the query :MEASURE:RISETIME? you must follow that query with an input statement. In HP BASIC, this is usually done with an ENTER statement immediately followed by a variable name. This statement reads the result of the query and places the result in a specified variable.

> Sending another command or query before reading the result of a query causes the output buffer to be cleared and the current response to be lost. This also generates a query interrupted error in the error queue.

Program Header Options

Program headers can be sent using any combination of uppercase or lowercase ASCII characters. Instrument responses, however, are always returned in uppercase.

Program command and query headers may be sent in either long form (complete spelling), short form (abbreviated spelling), or any combination of long form and short form.

TIMEBASE:DELAY 1US - long form

TIM:DEL 1US - short form

> Programs written in long form are easily read and are almost self-documenting. The short form syntax conserves the amount of controller memory needed for program storage and reduces the amount of I/O activity.

The rules for the short form syntax are shown in the chapter, "Programming Conventions."

Program Data Syntax Rules

Program data is used to convey a variety of types of parameter information related to the command header. At least one space must separate the command header or query header from the program data.

<program mnemonic><separator><data><terminator>

When a program mnemonic or query has multiple program data a comma separates sequential program data.

<program mnemonic><separator><data>,<data><terminator>

For example, :MEASURE:TVOLT 1.0V,2 has two program data: 1.0V and 2.

There are two main types of program data that are used in commands: character and numeric program data.

Character Program Data

Character program data is used to convey parameter information as alpha or alphanumeric strings. For example, the :TIMEBASE:REFERENCE command can be set to left, center, or right. The character program data in this case may be LEFT, CENTER, or RIGHT. :TIMEBASE:REFERENCE RIGHT sets the time base reference to right.

The available mnemonics for character program data are always included with the instruction's syntax definition. When sending commands, either the long form or short form (if one exists) may be used. Upper-case and lower-case letters may be mixed freely. When receiving responses, upper-case letters are used exclusively.

Numeric Program Data

Some command headers require program data to be expressed numerically. For example, :TIMEBASE:RANGE requires the desired full scale range to be expressed numerically.

For numeric program data, you have the option of using exponential notation or using suffix multipliers to indicate the numeric value. The following numbers are all equal:

28 = 0.28E2 = 280E-1 = 28000m = 0.028K = 28E-3K.

When a syntax definition specifies that a number is an integer, that means that the number should be whole. Any fractional part would be ignored, truncating the number. Numeric data parameters that accept fractional values are called real numbers. For more information see the chapter, "Interface Functions."

All numbers are expected to be strings of ASCII characters. Thus, when sending the number 9, you would send a byte representing the ASCII code for the character "9" (which is 57). A three-digit number like 102 would take up three bytes (ASCII codes 49, 48, and 50). This is taken care of automatically when you include the entire instruction in a string.

Embedded Strings

Embedded strings contain groups of alphanumeric characters which are treated as a unit of data by the oscilloscope. For example, the line of text written to the advisory line of the instrument with the :SYSTEM:DSP command:

:SYSTEM:DSP "This is a message."

Embedded strings may be delimited with either single (') or double (") quotes. These strings are case-sensitive and spaces act as legal characters just like any other character.

Program Message Terminator

The program instructions within a data message are executed after the program message terminator is received. The terminator may be either an NL (New Line) character, an EOI (End-Or-Identify) asserted in the HP-IB interface, or a combination of the two. All three ways are equivalent. Asserting the EOI sets the EOI control line low on the last byte of the data message. The NL character is an ASCII linefeed (decimal 10).

> The NL (New Line) terminator has the same function as an EOS (End Of String) and EOT (End Of Text) terminator.

Selecting Multiple Subsystems

You can send multiple program commands and program queries for different subsystems on the same line by separating each command with a semicolon. The colon following the semicolon enables you to enter a new subsystem. For example:

<program mnemonic><data>;:<program mnemonic><data><terminator>

:CHANNEL1:RANGE 0.4;:TIMEBASE:RANGE 1

> Multiple commands may be any combination of compound and simple commands.

Programming Getting Started

The remainder of this chapter deals mainly with how to set up the instrument, how to retrieve setup information and measurement results, how to digitize a waveform, and how to pass data to the controller. Refer to the chapter , "Measure Subsystem" for information on sending measurement data to the instrument.

> The programming examples in this manual are written in HP BASIC 5.0 for an HP 9000 Series 200/300 Controller.

Initialization

To make sure the bus and all appropriate interfaces are in a known state, begin every program with an initialization statement. For example, HP BASIC provides a CLEAR command which clears the interface buffer:

CLEAR 707 ! initializes the interface of the instrument

When you are using HP-IB, CLEAR also resets the oscilloscope's parser. The parser is the program that reads in the instructions that you send.

After clearing the interface, initialize the instrument to a preset state:

OUTPUT 707;"*RST" ! initializes the instrument to a preset state.

> The actual commands and syntax for initializing the instrument are discussed in the chapter, "Common Commands."
>
> Refer to your controller manual and programming language reference manual for information on initializing the interface.

Autoscale

The AUTOSCALE feature of Hewlett-Packard digitizing oscilloscopes performs a very useful function on unknown waveforms by setting up the vertical channel, time base, and trigger level of the instrument.

The syntax for the autoscale function is:

:AUTOSCALE<terminator>

Setting Up the Instrument

A typical oscilloscope setup would set the vertical range and offset voltage, the horizontal range, delay time, delay reference, trigger mode, trigger level, and slope.

A typical example of the commands sent to the oscilloscope are:

```
:CHANNEL1:PROBE 10; RANGE 16;OFFSET 1.00<terminator>
:TIMEBASE:MODE NORMAL;RANGE 1E-3;DELAY 100E-6<terminator>
```

This example sets the timebase at 1 ms full-scale (100 μs/div) with delay of 100 μs. Vertical is set to 16 V full-scale (2 V/div) with center of screen at 1 V and probe attenuation of 10.

Example Program

This program demonstrates the basic command structure used to program the oscilloscope.

```
10    CLEAR 707       ! Initialize instrument interface
20    OUTPUT 707;"*RST"     !Initialize instrument to preset state
30    OUTPUT 707;":TIMEBASE:RANGE 5E-4"  ! Time base to 500 us
                                            full scale
40    OUTPUT 707;":TIMEBASE:DELAY 0"     ! Delay to zero
50    OUTPUT 707;":TIMEBASE:REFERENCE CENTER"   ! Display
                                                   reference at center
60    OUTPUT 707;":CHANNEL1:PROBE 10"    ! Probe attenuation to 10:1
70    OUTPUT 707;":CHANNEL1:RANGE 1.6"   ! Vertical range to 1.6 V full scale
80    OUTPUT 707;":CHANNEL1:OFFSET -.4"  ! Offset to -0.4
90    OUTPUT 707;":CHANNEL1:INPUT DC"    ! Coupling to DC
100   OUTPUT 707;":TRIGGER:MODE EDGE"    ! Edge triggering
110   OUTPUT 707;":TRIGGER:LEVEL chanl,-.4"     ! Trigger level to -0.4
120   OUTPUT 707;":TRIGGER:SLOPE POSITIVE"      ! Trigger on positive slope
130   OUTPUT 707;":ACQUIRE:TYPE NORMAL"  ! Normal acquisition
140   OUTPUT 707;":DISPLAY:GRATICULE FRAME"     ! Grid off
150   END
```

Program Overview

Line 10 initializes the instrument interface to a known state.
Line 20 initializes the instrument to a preset state.
Lines 30 through 50 set the time base mode to normal with the horizontal time at 500 μs full scale with 0 s of delay referenced at the center of the graticule.
Lines 60 through 90 set the vertical range to 1.6 volts full scale with center screen at −0.4 volts with 10:1 probe attenuation and DC coupling.
Lines 100 through 120 configures the instrument to trigger at −0.4 volts with normal triggering.
Line 130 configures the instrument for normal acquisition.
Line 140 turns the grid off.

Using the Digitize Command

The Digitize command is a macro that captures data satisfying the specifications set up by the acquire subsystem. When the digitize process is complete, the acquisition is stopped. The captured data can then be measured by the instrument or transferred to the controller for further analysis. The captured data consists of two parts: the waveform data record and the preamble.

After changing the oscilloscope configuration, the waveform buffers are cleared. Before doing a measurement, the Digitize command should be sent to ensure new data has been collected.

The DIGITIZE command can be sent without parameters for a higher throughput. Refer to the DIGITIZE command in the Root Level Commands chapter for details.

When the DIGITIZE command is sent to an instrument, the specified channel signal is digitized with the current ACQUIRE parameters. To obtain waveform data, you must specify the WAVEFORM parameters for the waveform data prior to sending the :WAVEFORM:DATA? query.

The number of data points comprising a waveform varies according to the number requested in the ACQUIRE subsystem. The ACQUIRE subsystem determines the number of data points, type of acquisition, and number of averages used by the DIGITIZE command. This allows you to specify exactly what the digitized information contains. The following program example shows a typical setup:

```
OUTPUT 707;":ACQUIRE:TYPE AVERAGE"<terminator>
OUTPUT 707;":ACQUIRE:COMPLETE 100"<terminator>
OUTPUT 707;":WAVEFORM:SOURCE CHANNEL1"<terminator>
OUTPUT 707;":WAVEFORM:FORMAT BYTE"<terminator>
OUTPUT 707;":ACQUIRE:COUNT 8"<terminator>
OUTPUT 707;":ACQUIRE:POINTS 500"<terminator>
OUTPUT 707;":DIGITIZE CHANNEL1"<terminator>
OUTPUT 707;":WAVEFORM:DATA?"<terminator>
```

This setup places the instrument into the averaged mode with eight averages. This means that when the DIGITIZE command is received, the command will execute until the signal has been averaged at least eight times.

After receiving the :WAVEFORM:DATA? query, the instrument will start passing the waveform information when addressed to talk.

Digitized waveforms are passed from the instrument to the controller by sending a numerical representation of each digitized point. The format of the numerical representation is controlled with the :WAVEFORM:FORMAT command and may be selected as BYTE, WORD, or ASCII.

The easiest method of entering a digitized waveform depends on data structures, available formatting and I/O capabilities. You must scale the integers to determine the voltage value of each point. These integers are passed starting with the leftmost point on the instrument's display. For more information, refer to chapter 18, "Waveform Subsystem."

When using HP-IB, a digitize operation may be aborted by sending a Device Clear over the bus (CLEAR 707).

Receiving Information from the Instrument

After receiving a query (command header followed by a question mark), the instrument interrogates the requested function and places the answer in its output queue. The answer remains in the output queue until it is read or another command is issued. When read, the answer is transmitted across the interface to the designated listener (typically a controller). The input statement for receiving a response message from an instrument's output queue typically has two parameters; the device address, and a format specification for handling the response message. For example, to read the result of the query command :CHANNEL1:COUPLING? you would execute the HP BASIC statement:

ENTER <device address> ;Setting$

where <device address> represents the address of your device. This would enter the current setting for the channel one coupling in the string variable Setting$.

All results for queries sent in a program message must be read before another program message is sent. For example, when you send the query :MEASURE:RISETIME?, you must follow that query with an input statement. In HP BASIC, this is usually done with an ENTER statement.

Sending another command before reading the result of the query causes the output buffer to be cleared and the current response to be lost. This also causes an error to be placed in the error queue.

Executing an input statement before sending a query causes the controller to wait indefinitely.

The format specification for handling response messages is dependent on both the controller and the programming language.

String Variable Example

The output of the instrument may be numeric or character data depending on what is queried. Refer to the specific commands for the formats and types of data returned from queries.

For the example programs, assume that the device being programmed is at device address 707. The actual address varies according to how you have configured the bus for your own application.

In HP BASIC 5.0, string variables are case sensitive and must be expressed exactly the same each time they are used. The following example shows the data being returned to a string variable:

```
10 DIM Rang$[30]
20 OUTPUT 707;":CHANNEL1:RANGE?"
30 ENTER 707;Rang$
40 PRINT Rang$
50 END
```

After running this program, the controller displays:

+8.00000E-01

In this example, the oscilloscope is set to 100 mV/division given this output value.

Numeric Variable Example

The following example shows the data being returned to a numeric variable:

```
10 OUTPUT 707;":CHANNEL1:RANGE?"
20 ENTER 707;Rang
30 PRINT Rang
40 END
```

After running this program, the controller displays:

.8

Definite-Length Block Response Data

Definite-length block response data allows any type of device-dependent data to be transmitted over the system interface as a series of 8-bit binary data bytes. This is particularly useful for sending large quantities of data or 8-bit extended ASCII codes. The syntax is a pound sign (#) followed by a non-zero digit representing the number of digits in the decimal integer. After the non-zero digit is the decimal integer that states the number of 8-bit data bytes being sent. This is followed by the actual data.

For example, for transmitting 4000 bytes of data, the syntax would be:

#44000 <4000 bytes of data> <terminator>

The "4" represents the number of digits that follow, and "4000" represents the number of bytes to be transmitted.

Multiple Queries

You can send multiple queries to the instrument within a single program message, but you must also read them back within a single program message. This can be accomplished by either reading them back into a string variable or into multiple numeric variables. For example, you could read the result of the query :TIMEBASE:RANGE?;DELAY? into the string variable Results$ with the command:

ENTER 707;Results$

When you read the result of multiple queries into string variables, each response is separated by a semicolon. For example, the response of the query :TIMEBASE:RANGE?;DELAY? would be:

<range_value>; <delay_value>

Use the following program message to read the query :TIMEBASE:RANGE?;DELAY? into multiple numeric variables:

ENTER 707;Result1,Result2

Instrument Status

Status registers track the current status of the instrument. By checking the instrument status, you can find out whether an operation has been completed, whether the instrument is receiving triggers, and more. The chapter on "Status Reporting" explains how to check the status of the instrument.

2

Interface Functions

Interface Functions

The interface functions deal with general bus management issues, as well as messages which can be sent over the bus as bus commands. In general, these functions are defined by IEEE 488.1.

HP-IB Interface Connector

The oscilloscope is equipped with an HP-IB interface connector on the rear panel. This allows direct connection to an HP-IB compatible printer or external controller. An external HP-IB compatible device can be connected to the oscilloscope by installing an HP-IB cable between the two units. Finger tighten the captive screws on both ends of the HP-IB cable to avoid accidently disconnecting the cable during operation.

Up to fifteen HP-IB compatible instruments (including a controller) can be interconnected in a system by stacking (piggy-backing) connectors. This allows the instruments to be connected in virtually any configuration desired, as long as there is a path from the controller to every device operating on the bus.

CAUTION

Avoid stacking more than three or four cables on any one connector. Multiple connectors produce leverage that can damage a connector mounting.

HP-IB Default Startup Conditions

The following default HP-IB conditions are established during power-up.

- HP-IB local mode is active.
- Local lockout is cleared.
- The Request Service (RQS) bit in the status byte register is set to zero.
- All event registers, the Standard Event Status Enable Register, Service Request Enable Register, and the Status Byte Register are cleared.

Interface Capabilities

The interface capabilities of this oscilloscope, as defined by IEEE 488.1, are listed in the following table.

Table 2-1 **Interface Capabilities**

Code	Interface Function	Capability
SH1	Source Handshake	Full Capability
AH1	Acceptor Handshake	Full Capability
T5	Talker	Basic Talker/Serial Poll/Talk Only Mode/ Unaddress if Listen Address (MLA)
L4	Listener	Basic Listener/ Unaddresses if Talk Address (MTA)
SR1	Service Request	Full Capability
RL1	Remote Local	Complete Capability
PP1	Parallel Poll	Remote Configuration
DC1	Device Clear	Full Capability
DT1	Device Trigger	Full Capability
C0	Controller	No Capability
E2	Driver Electronics	Tri State (1 MB/SEC MAX)

Command and Data Concepts

The HP-IB has two modes of operation: command mode and data mode. The bus is in the command mode when the Attention (ATN) control line is true. The command mode is used to send talk and listen addresses and various bus commands such as group execute trigger (GET).

The bus is in the data mode when the ATN line is false. The data mode is used to convey device-dependent messages across the bus. The device-dependent messages include all of the oscilloscope specific commands, queries, and responses found in this manual including instrument status information.

Addressing

The oscilloscope is always in the addressed (talk/listen) mode from the HP-IB menu of the front panel of the oscilloscope. The HP-IB menu is selected by pressing the Utility key on the front panel, then selecting the HP-IB softkey.

Addressed mode is used when the instrument operates in conjunction with a controller. When the instrument is in the addressed mode, the following is true:

- Each device on the HP-IB resides at a particular address, ranging from 0 to 30.
- The active controller specifies which devices talk and which listen.
- An instrument may be talk addressed, listen addressed, or unaddressed by the controller.

If the controller addresses an instrument to talk, the instrument remains configured to talk until it receives an interface clear message (IFC), another instrument's talk address (OTA), its own listen message (MLA), or a universal untalk command (UNT).

If the controller addresses an instrument to listen, the instrument remains configured to listen until it receives an interface clear message (IFC), its own talk address (MTA), or a universal unlisten command (UNL).

Communicating Over the Bus

Device addresses are sent by the controller in the command mode to specify who talks and who listens. Since HP-IB can address multiple devices through the same interface card, the device address passed with the program message must include not only the correct interface select code, but also the correct instrument address.

Device Address = (Interface Select Code * 100) + (Instrument Address)

> The examples in this manual assume that the oscilloscope is at device address 707.

Interface Select Code

Each interface card has a unique interface select code. This code is used by the controller to direct commands and communications to the proper interface. The default is typically "7" for HP-IB controllers.

Instrument Address

Each instrument on the HP-IB must have a unique instrument address between decimal 0 and 30. This instrument address is used by the controller to direct commands and communications to the proper instrument on an interface. The default is typically "7" for this oscilloscope. This address can be changed in the HP-IB menu of the Utility menu of the oscilloscope.

> Address 21 is usually reserved for the Computer interface Talk/Listen address and should not be used as an instrument address.

Remote, Local, and Local Lockout

The remote, local, and local lockout modes are used for various degrees of front-panel control while a program is running.

The instrument accepts and executes bus commands while in the local mode with all front-panel controls active.

The instrument is placed in the remote mode when the controller sets the Remote Enable (REN) bus control line true and addresses the instrument to listen. In the remote mode, all controls except the power switch and the front-panel LOCAL key are entirely locked out. Local control can only be restored by the controller or by pressing the front-panel LOCAL key.

> Cycling the power also restores all front-panel controls (local mode), but this also resets certain HP-IB states.

The Local Lockout command (LLO) disables all front-panel controls including the LOCAL key. The only active control is the power switch. This prevents undesired or accidental front-panel control which could result in data or settings being changed. The instrument accepts the Local Lockout command whether the instrument is addressed in the remote or local mode. The instrument is returned to the local mode by either setting the REN line false, or by sending the go-to-local command (GTL) to the instrument.

Bus Commands

The following commands are IEEE 488.1 bus commands (ATN true). IEEE 488.2 defines many of the actions that are taken when these commands are received by the instrument.

Device Clear

The device clear (DCL) and selected device clear (SDC) commands clear the input buffer and output queue, reset the parser, and clear any pending commands. If either of these commands is sent during a digitize operation, the digitize operation is aborted.

Group Execute Trigger

The group execute trigger (GET) command arms the trigger. This is the same action produced by sending the RUN command.

Interface Clear

The interface clear (IFC) command halts all bus activity. This includes unaddressing all listeners and the talker, disabling serial poll on all devices, and returning control to the system controller.

Status Messages

When the instrument is in the remote mode, the Remote message is displayed on the oscilloscope screen.

3

Message Communication and System Functions

Message Communication and System Functions

This chapter describes the operation of instruments that operate in compliance with the IEEE 488.2 (syntax) standard. It is intended to give you enough basic information about the IEEE 488.2 Standard to successfully program the instrument. You can find additional detailed information about the IEEE 488.2 Standard in ANSI/IEEE Std 488.2-1987, *"IEEE Standard Codes, Formats, Protocols, and Common Commands."*

This instrument series is designed to be compatible with other Hewlett-Packard IEEE 488.2 compatible instruments. Instruments that are compatible with IEEE 488.2 must also be compatible with IEEE 488.1 (HP-IB bus standard); however, IEEE 488.1 compatible instruments may or may not conform to the IEEE 488.2 standard. The IEEE 488.2 standard defines the message exchange protocols by which the instrument and the controller will communicate. It also defines some common capabilities that are found in all IEEE 488.2 instruments. This chapter also contains a few items which are not specifically defined by IEEE 488.2, but which deal with message communication or system functions.

Protocols

The message exchange protocols of IEEE 488.2 define the overall scheme used by the controller and the instrument to communicate. This includes defining when it is appropriate for devices to talk or listen, and what happens when the protocol is not followed.

Functional Elements

Before proceeding with the description of the protocol, a few system components should be understood.

Input Buffer The input buffer of the instrument is the memory area where commands and queries are stored prior to being parsed and executed. It allows a controller to send a string of commands, which could take some time to execute, to the instrument, and then proceed to talk to another instrument while the first instrument is parsing and executing commands.

Output Queue The output queue of the instrument is the memory area where all output data (<response messages>) are stored until read by the controller.

Parser The instrument's parser is the component that interprets the commands sent to the instrument and decides what actions should be taken. "Parsing" refers to the action taken by the parser to achieve this goal. Parsing and executing of commands begins when either the instrument recognizes a <program message terminator> (defined later in this chapter) or the input buffer becomes full. If you wish to send a long sequence of commands to be executed and then talk to another instrument while they are executing, you should send all the commands before sending the <program message terminator>.

Protocol Overview

The instrument and controller communicate using <program message>s and <response message>s. These messages serve as the containers into which sets of program commands or instrument responses are placed. <program message>s are sent by the controller to the instrument, and <response message>s are sent from the instrument to the controller in response to a query message. A <query message> is defined as being a <program message> that contains one or more queries. The instrument will only talk when it has received a valid query message, and therefore has something to

say. The controller should only attempt to read a response after sending a complete query message, but before sending another <program message>. The basic rule to remember is that the instrument will only talk when prompted to, and it then expects to talk before being told to do something else.

Protocol Operation

When the instrument is turned on, the input buffer and output queue are cleared, and the parser is reset to the root level of the command tree.

The instrument and the controller communicate by exchanging complete <program message>s and <response message>s. This means that the controller should always terminate a <program message> before attempting to read a response. The instrument will terminate <response message>s except during a hardcopy output.

If a query message is sent, the next message passing over the bus should be the <response message>. The controller should always read the complete <response message> associated with a query message before sending another <program message> to the same instrument.

The instrument allows the controller to send multiple queries in one query message. This is referred to as sending a "compound query." As noted later in this chapter, multiple queries in a query message are separated by semicolons. The responses to each of the queries in a compound query will also be separated by semicolons.

Commands are executed in the order they are received.

Protocol Exceptions

If an error occurs during the information exchange, the exchange may not be completed in a normal manner. The following are some of the protocol exceptions.

Command Error A command error is reported if the instrument detects a syntax error or an unrecognized command header.

Execution Error An execution error is reported if a parameter is found to be out of range, or if the current settings do not allow execution of a requested command or query.

Device-specific Error A device-specific error is reported if the instrument is unable to execute a command for a strictly device dependent reason.

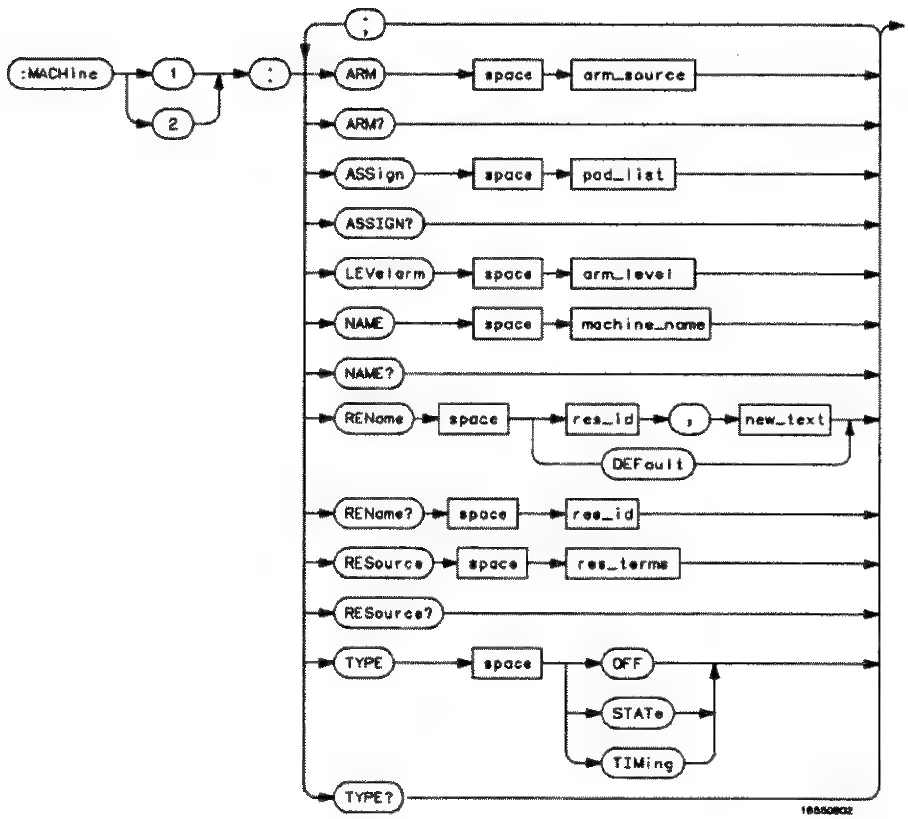
Query Error A query error is reported if the proper protocol for reading a query is not followed. This includes the interrupted and unterminated conditions described in the following paragraphs.

Syntax Diagrams

The example syntax diagrams in this chapter are similar to the syntax diagrams in the IEEE 488.2 specification. Commands and queries are sent to the instrument as a sequence of data bytes. The allowable byte sequence for each functional element is defined by the syntax diagram that is shown.

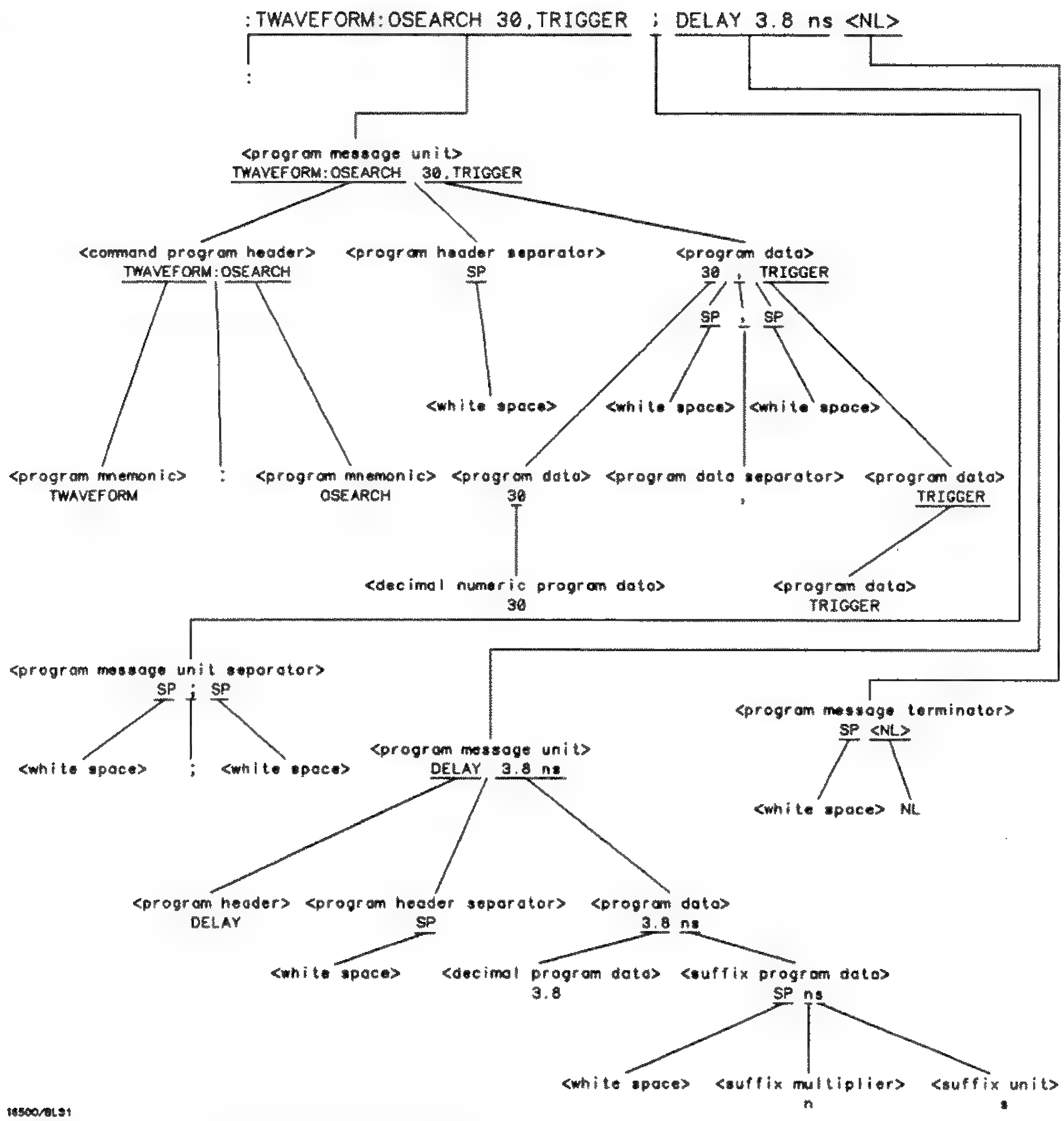
The allowable byte sequence can be determined by following a path in the syntax diagram. The proper path through the syntax diagram is any path that follows the direction of the arrows. If there is a path around an element, that element is optional. If there is a path from right to left around one or more elements, that element or those elements may be repeated as many times as desired.

Figure 3-1



Example Syntax Diagram

Figure 3-2



<program message> Parse Tree

Syntax Overview

This overview is intended to give a quick glance at the syntax defined by IEEE 488.2. It will help you understand many of the things about the syntax you need to know.

IEEE 488.2 defines the blocks used to build messages that are sent to the instrument. A whole string of commands can therefore be broken up into individual components.

Figure 3-1 is an example syntax diagram and figure 3-2 shows a breakdown of an example <program message> in the parse tree. There are a few key items to notice:

- A semicolon separates commands from one another. Each <program message unit> serves as a container for one command. The <program message unit>s are separated by a semicolon.
- A <program message> is terminated by a <NL> (new line). The recognition of the <program message terminator>, or <PMT>, by the parser serves as a signal for the parser to begin execution of commands. The <PMT> also affects instrument command tree traversal.
- Multiple data parameters are separated by a comma.
- The first data parameter is separated from the header with one or more spaces.
- The header MACHINE1:ASSIGN 2,3 is an example of a compound header. It places the parser in the machine subsystem until the <NL> is encountered.
- A colon preceding the command header returns you to the top of the parser tree.

Upper/Lower Case Equivalence

Upper and lower case letters are equivalent. The mnemonic `SINGLE` has the same semantics as the mnemonic `single`.

<white space>

<white space> is defined to be one or more characters from the ASCII set of 0 - 32 decimal, excluding 10 decimal (NL). <white space> is used by several instrument listening components of the syntax. It is usually optional, and can be used to increase the readability of a program.

Suffix Multiplier The suffix multipliers that the instrument will accept are shown in table 3-1.

Table 3-1 <suffix mult>

Value	Mnemonic
1E18	EX
1E15	PE
1E12	T
1E9	G
1E6	MA
1E3	K
1E-3	M
1E-6	U
1E-9	N
1E-12	P
1E-15	F
1E-18	A

Suffix Unit The suffix units that the instrument will accept are shown in table 3-2.

Table 3-2

<suffix unit>

Suffix	Referenced Unit
V	Volt
S	Second

4

Status Reporting

Status Reporting

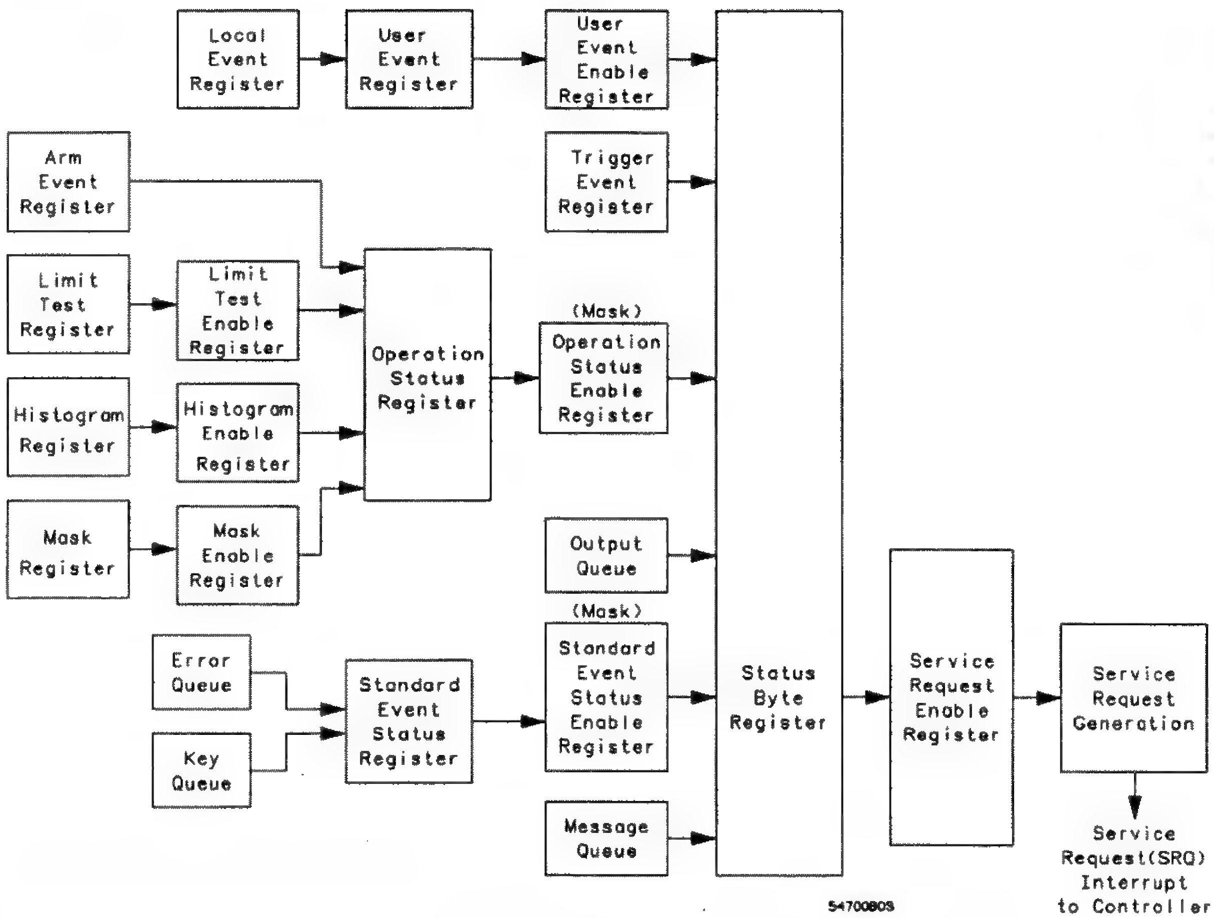
Figure 4-1 is an overview of the oscilloscope's status reporting structure. The status reporting structure allows monitoring specified events in the oscilloscope. The ability to monitor and report these events allows determination of such things as the status of an operation, the availability and reliability of the measured data, and more.

- To monitor an event, first clear the event, then enable the event. All of the events are cleared when you initialize the instrument.
- To generate a service request (SRQ) interrupt to an external controller, enable at least one bit in the Status Byte Register.

The Status Byte Register, the Standard Event Status Register group, and the Output Queue are defined as the Standard Status Data Structure Model in IEEE 488.2-1987.

IEEE 488.2 defines data structures, commands, and common bit definitions for status reporting. There are also instrument-defined structures and bits.

Figure 4-1



Status Reporting Overview Block Diagram

The status reporting structure consists of the registers in figure 4-1.

Table 4-1 is a list of the bit definitions for the bit in the status reporting data structure.

Table 4-1

Status Reporting Bit Definition

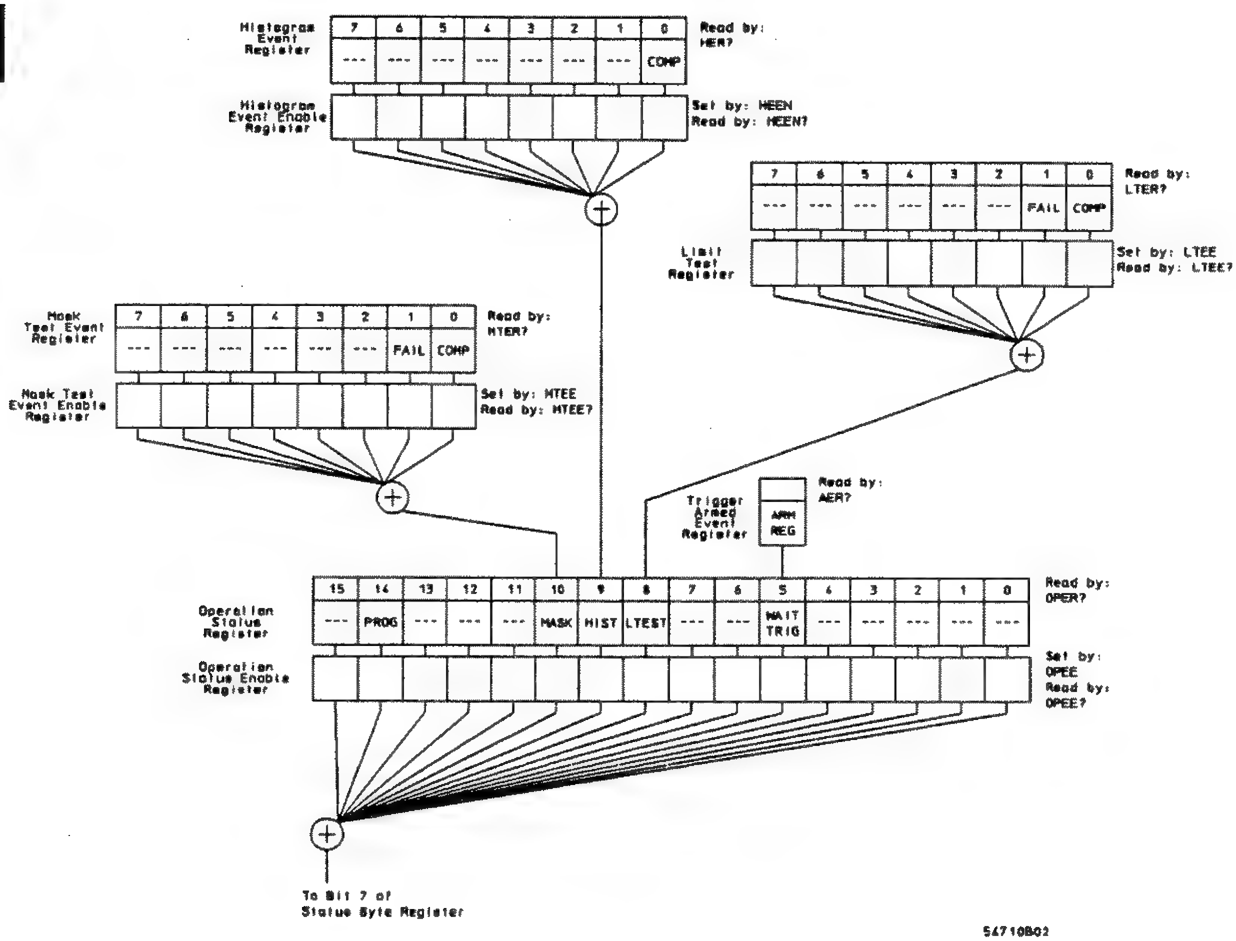
Bit	Description	Definition
PON	Power On	Indicates power is turned on.
URQ	User Request	Indicates whether a front-panel key has been pressed.
CME	Command Error	Indicates whether the parser detected an error.
EXE	Execution Error	Indicates whether a parameter was out of range, or inconsistent with the current settings.
DDE	Device Dependent Error	Indicates whether the device was unable to complete an operation for device dependent reasons.
QYE	Query Error	Indicates if the protocol for queries has been violated.
RQL	Request Control	Indicates whether the device is requesting control.
OPC	Operation Complete	Indicates whether the device has completed all pending operations.
OPER	Operation Status Register	Indicates if any of the enabled conditions in the Operation Status Register have occurred.
RQS	Request Service	Indicates that the device is requesting service.
MSS	Master Summary Status	Indicates whether a device has a reason for requesting service.
ESB	Event Status Bit	Indicates if any of the enabled conditions in the Standard Event Status Register have occurred.
MAV	Message Available	Indicates if there is a response in the output queue.
MSG	Message	Indicates whether an advisory has been displayed.
USR	User Event Register	Indicates if any of the enabled conditions have occurred in the User Event Register.
TRG	Trigger	Indicates whether a trigger has been received.
LCL	Local	Indicates if a remote-to-local transition occurs.
FAIL	Fail	Indicates that the specified test has failed.
COMP	Complete	Indicates that the specified test has completed.
LTEST	Limit Test	Indicates if any of the enabled conditions have occurred in the Limit Test Register.
MTEST	Mask Test	Indicates if any of the enabled conditions have occurred in the Mask Test Register.
HIST	Histogram	Indicates if any of the enabled conditions have occurred in the Histogram Register.
WAIT TRIG	Wait for Trigger	Indicates instrument is armed and ready for trigger.

Status Reporting Data Structures

Figure 4-2 brings together the different status reporting data structures mentioned in this chapter and shows how they work together. To make it possible for any of the Standard Event Status Register bits to generate a summary bit, the bits must be enabled. These bits are enabled by using the *ESE common command to set the corresponding bit in the Standard Event Status Enable Register.

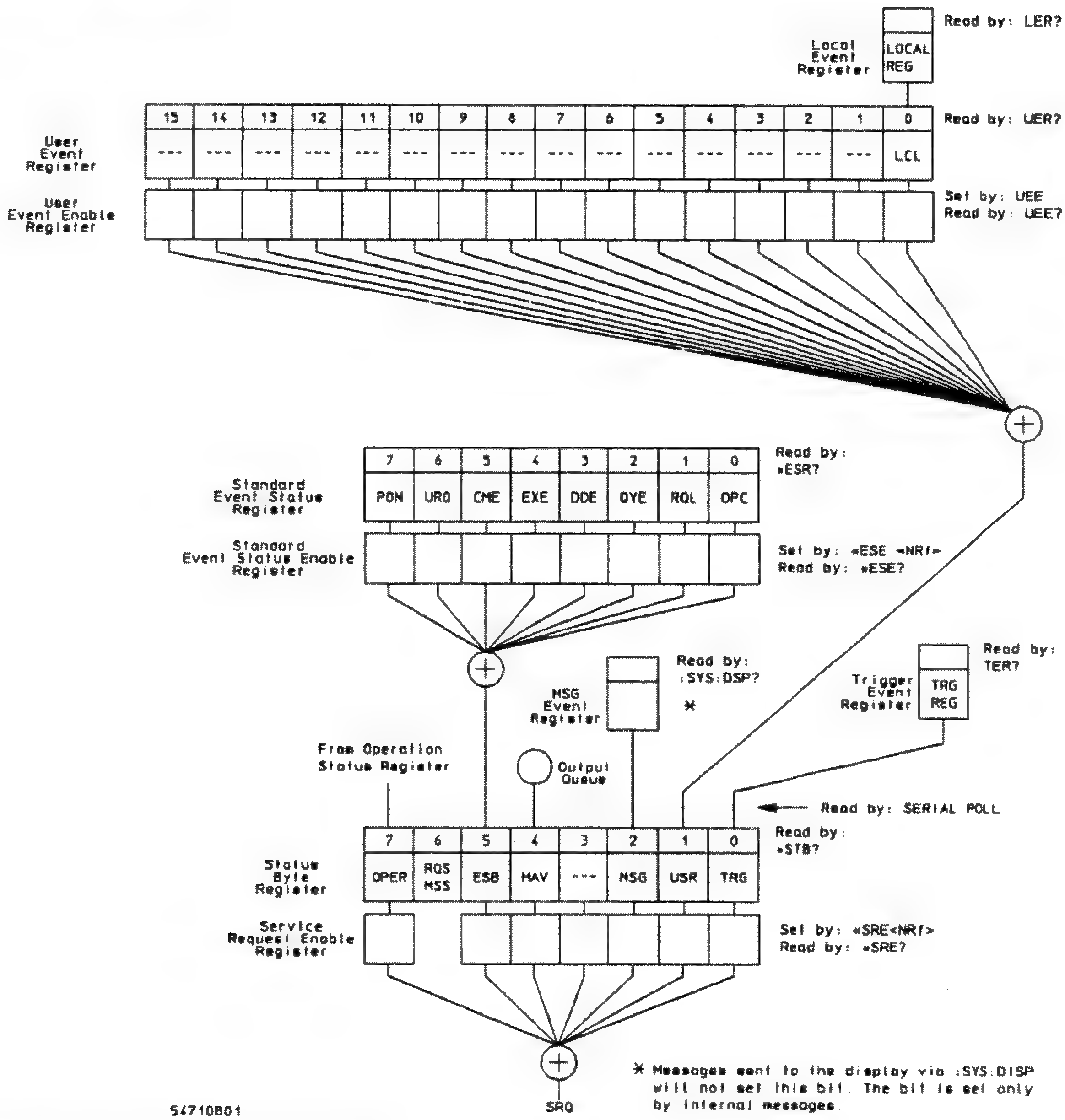
To generate a service request (SRQ) interrupt to an external controller, at least one bit in the Status Byte Register must be enabled. These bits are enabled by using the *SRE common command to set the corresponding bit in the Service Request Enable Register. These enabled bits can then set RQS and MSS (bit 6) in the Status Byte Register.

Figure 4-2



Status Reporting Data Structures

Figure 4-2 (continued)



Status Reporting Data Structures (continued)

Status Byte Register

The Status Byte Register is the summary-level register in the status reporting structure. It contains summary bits that monitor activity in the other status registers and queues. The Status Byte Register is a live register. That is, its summary bits are set and cleared by the presence and absence of a summary bit from other event registers or queues.

If the Status Byte Register is to be used with the Service Request Enable Register to set bit 6 (RQS/MSS) and to generate an SRQ, at least one of the summary bits must be enabled, then set. Also, event bits in all other status registers must be specifically enabled to generate the summary bit that sets the associated summary bit in the Status Byte Register.

The Status Byte Register can be read using either the *STB? Common Command or the HP-IB serial poll command. Both commands return the decimal-weighted sum of all set bits in the register. The difference between the two methods is that the serial poll command reads bit 6 as the Request Service (RQS) bit and clears the bit which clears the SRQ interrupt. The *STB? command reads bit 6 as the Master Summary Status (MSS) and does not clear the bit or have any affect on the SRQ interrupt. The value returned is the total bit weights of all of the bits that are set at the present time.

The use of bit 6 can be confusing. This bit was defined to cover all possible computer interfaces, including a computer that could not do a serial poll. The important point to remember is that, if you are using an SRQ interrupt to an external computer, the serial poll command clears bit 6. Clearing bit 6 allows the oscilloscope to generate another SRQ interrupt when another enabled event occurs.

No other bits in the Status Byte Register are cleared by either the *STB? query or the serial poll, except the Message Available bit (bit 4). If there are no other messages in the Output Queue, bit 4 (MAV) can be cleared as a result of reading the response to the *STB? command.

If bit 4 (weight = 16) and bit 5 (weight = 32) are set, the program prints the sum of the two weights. Since these bits were not enabled to generate an SRQ, bit 6 (weight = 64) is not set.

Example 1

The following example uses the *STB? query to read the contents of the oscilloscopes Status Byte Register when none of the register's summary bits are enabled to generate an SRQ interrupt.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF;*STB?"     !Turn headers off
20  ENTER 707;Result       !Place result in a numeric variable
30  PRINT Result      !Print the result
40  End
```

The next program prints 112 and clears bit 6 (RQS) of the Status Byte Register. The difference in the decimal value between this example and the previous one is the value of bit 6 (weight = 64). Bit 6 is set when the first enabled summary bit is set and is cleared when the Status Byte Register is read by the serial poll command.

Example 2

The following example uses the HP BASIC serial poll (SPOLL) command to read the contents of the oscilloscopes Status Byte Register.

```
10  Result = SPOLL(707)
20  PRINT Result
30  END
```

> Serial polling is the preferred method to read the contents of the Status Byte Register because it resets bit 6 and allows the next enabled event that occurs to generate a new SRQ interrupt.

Service Request Enable Register

Setting the Service Request Enable Register bits enable corresponding bits in the Status Byte Register. These enabled bits can then set RQS and MSS (bit 6) in the Status Byte Register.

Bits are set in the Service Request Enable Register using the *SRE command and the bits that are set are read with the *SRE? query.

Refer to figure 4-2.

Example

The following example sets bit 4 (MAV) and bit 5 (ESB) in the Service Request Enable Register.

```
OUTPUT 707;"*SRE 48"
```

This example uses the parameter "48" to enable the oscilloscope to generate an SRQ interrupt under the following conditions:

- When one or more bytes in the Output Queue set bit 4 (MAV).
- When an enabled event in the Standard Event Status Register generates a summary bit that sets bit 5 (ESB).

Trigger Event Register (TRG)

This register sets the TRG bit in the status byte when a trigger event occurs.

The TRG event register stays set until it is cleared by reading the register or using the *CLS command. If your application needs to detect multiple triggers, the TRG event register must be cleared after each one.

If you are using the Service Request to interrupt a program or controller operation when the trigger bit is set, then you must clear the event register after each time it has been set.

Standard Event Status Register

The Standard Event Status Register (SESR) monitors the following oscilloscope status events:

- PON - Power On,
- URQ - User Request,
- CME - Command Error,
- EXE - Execution Error,
- DDE - Device Dependent Error,
- QYE - Query Error,
- RQC - Request Control, and
- OPC - Operation Complete.

When one of these events occur, the event sets the corresponding bit in the register. If the bits are enabled in the Standard Event Status Enable Register, the bits set in this register generate a summary bit to set bit 5 (ESB) in the Status Byte Register.

The contents of the Standard Event Status Register can be read and the register cleared by sending the *ESR? query. The value returned is the total bit weights of all of the bits that are set at the present time.

Example

The following example uses the *ESR query to read the contents of the Standard Event Status Register.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"      !Turn headers off
20  OUTPUT 707;"*ESR?"
30  ENTER 707;Result      !Place result in a numeric variable
40  PRINT Result      !Print the result
50  End
```

If bit 4 (weight = 16) and bit 5 (weight = 32) are set, the program prints the sum of the two weights.

Standard Event Status Enable Register

To make it possible for any of the Standard Event Status Register (SESR) bits to be able to generate a summary bit, first enable the bit. Enable the bit by using the *ESE (Event Status Enable)common command to set the corresponding bit in the Standard Event Status Enable Register.

Set bits are read with the *ESE? query.

Example

For example, suppose your application requires an interrupt whenever any type of error occurs. The error related bits in the Standard Event Status Register are bits 2 through 5. The sum of the decimal weights of these bits is 60. Therefore, you can enable any of these bits to generate the summary bit by sending:

```
OUTPUT 707;"*ESE 60"
```

Whenever an error occurs, it sets one of these bits in the Standard Event Status Register. Because the bits are all enabled, a summary bit is generated to set bit 5 (ESB) in the Status Byte Register.

If bit 5 (ESB) in the Status Byte Register is enabled (via the *SRE command), an SRQ service request interrupt is sent to the external computer.

> Standard Event Status Register bits that are not enabled still respond to their corresponding conditions (that is, they are set if the corresponding event occurs). However, because they are not enabled, they do not generate a summary bit to the Status Byte Register.

User Event Register (UER)

This register hosts the LCL bit (bit 0) from the Local Event Register. The other 15 bits are reserved. You can read and clear this register using the UER? query. This register is enabled with the UEE command. For example, if you want to enable the LCL bit, you send a mask value of 1 with the UEE command; otherwise, send a mask value of 0.

Local Event Register (LCL)

This register sets the LCL bit in the User Event Register and the USR bit (bit 1) in the status byte. It indicates a remote-to-local transition has occurred. The LER? query is used to read and to clear this register.

Operation Status Register (OPR)

This register hosts the WAIT TRIG bit (bit 5), the LTEST bit (bit 8), the HIST bit (bit 9), the MASK bit (bit 10), and the PROG bit (bit 14).

The WAIT TRIG bit is set by the Trigger Armed Event Register and indicates that the trigger is armed.

The LTEST bit is set when a limit test fails or is completed and sets the corresponding FAIL or COMP bits in the Limit Test Event Register.

The HIST bit is set when the COMP bit is set in the Histogram Event Register, indicating that the histogram measurement has satisfied the specified completion criteria.

The MASK bit is set when the Mask Test either fails specified conditions or satisfies its completion criteria, setting the corresponding FAIL or COMP bits in the Mask Test Event Register.

The PROG bit is reserved for future use.

If any of these bits are set, the OPER bit (bit 7) of the Status Byte Register is set. The Operation Status Register is read and cleared with the OPER? query. The register output is enabled or disabled using the mask value supplied with the OPEE command.

Limit Test Event Register (LTER)

Bit 0 (COMP) of the Limit Test Event Register is set when the Limit Test completes. The Limit Test completion criteria are set by the LTESt:RUN command.

Bit 1 (FAIL) of the Limit Test Event Register is set when the Limit Test fails. Failure criteria for the Limit Test are defined by the LTESt:FAIL command.

The Limit Test Event Register is read and cleared with the LTER? query.

When either the COMP or FAIL bits are set, they in turn set the LTEST bit (bit 8) of the Operation Status Register. You can mask the COMP and FAIL bits, thus preventing them from setting the LTEST bit, by defining a mask using the LTEE command.

Enable	Mask Value
Block COMP and FAIL	0
Enable COMP, block FAIL	1
Enable FAIL, block COMP	2
Enable COMP and FAIL	3

Mask Test Event Register

Bit 0 (COMP) of the Mask Test Event Register is set when the Mask Test completes. The Mask Test completion criteria are set by the MTESt:RUMode command.

Bit 1 (FAIL) of the Mask Test Event Register is set when the Mask Test fails. This will occur whenever any sample is recorded within any polygon defined in the mask.

The Mask Test Event Register is read and cleared with the MTER? query.

When either the COMP or FAIL bits are set, they in turn set the MASK bit (bit 10) of the Operation Status Register. You can mask the COMP and FAIL bits, thus preventing them from setting the MASK bit, by defining a mask using the MTEE command.

Enable	Mask Value
Block COMP and FAIL	0
Enable COMP, block FAIL	1
Enable FAIL, block COMP	2
Enable COMP and FAIL	3

Histogram Event Register

Bit 0 (COMP) of the Histogram Event Register is set when the Histogram completes. The Histogram completion criteria are set by the HISTogram:RUNTil command. The Histogram Event Register is read and cleared with the HER? query.

When the COMP bit is set, it in turn sets the HIST bit (bit 9) of the Operation Status Register. Results from the Histogram Register can be masked by using the HEEN command to set the Histogram Event Enable Register to the value 0. You enable the COMP bit by setting the mask value to 1.

Arm Event Register (ARM)

This register sets bit 5 (Wait Trig bit) in the Operation Status Register and the OPER bit (bit 7) in the Status Byte Register when the instrument becomes armed.

The ARM event register stays set until it is cleared by reading the register with the AER? query or using the *CLS command. If your application needs to detect multiple triggers, the ARM event register must be cleared after each one.

If you are using the Service Request to interrupt a program or controller operation when the trigger bit is set, then you must clear the event register after each time it has been set.

Error Queue

As errors are detected, they are placed in an error queue. This queue is first in, first out. If the error queue overflows, the last error in the queue is replaced with error -350, "Queue overflow." Any time the queue overflows, the least recent errors remain in the queue, and the most recent error is discarded. The length of the oscilloscope's error queue is 30 (29 positions for the error messages, and 1 position for the "Queue overflow" message).

The error queue is read with the SYSTEM:ERROR? query. Executing this query reads and removes the oldest error from the head of the queue, which opens a position at the tail of the queue for a new error. When all the errors have been read from the queue, subsequent error queries return 0, "No error."

The error queue is cleared when any of the following items occur:

- When the instrument is powered up.
- When the instrument receives the *CLS common command.
- When the last item is read from the error queue.

For more information on reading the error queue, refer to the SYSTEM:ERROR? query in the System Commands chapter. For a complete list of error messages, refer to the chapter, "Error Messages."

Output Queue

The output queue stores the oscilloscope-to-controller responses that are generated by certain instrument commands and queries. The output queue generates the Message Available summary bit when the output queue contains one or more bytes. This summary bit sets the MAV bit (bit 4) in the Status Byte Register.

The output queue may be read with the HP Basic ENTER statement.

Message Queue

The message queue contains the text of the last message written to the advisory line on the screen of the oscilloscope. The length of the oscilloscope's message queue is 1. The queue is read with the SYSTEM:DSP? query. Note that messages sent with the SYSTem:DSP command do not set the MSG status bit in the Status Byte Register.

Key Queue

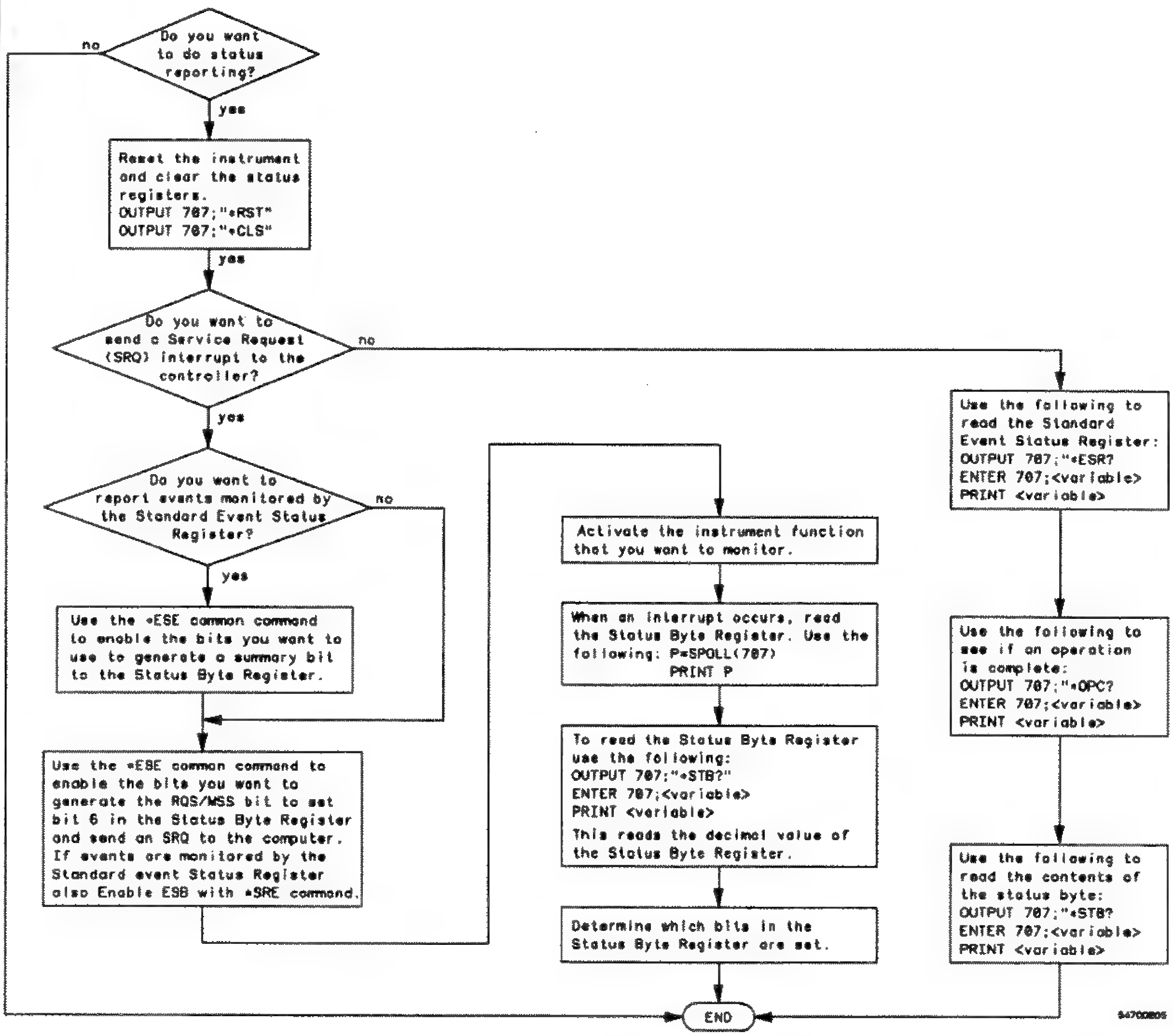
The key queue contains the key codes for the last 10 keys pressed on the front panel. This queue is first in, first out. If the key queue overflows, the oldest key codes are discarded as additional keys are pressed. The key queue is read with the SYSTEM:KEY? query.

Clearing Registers and Queues

The *CLS common command clears all event registers and all queues except the output queue. If *CLS is sent immediately following a program message terminator, the output queue is also cleared.

Figure 4-3

Status Reporting Decision Chart



5

Programming Syntax

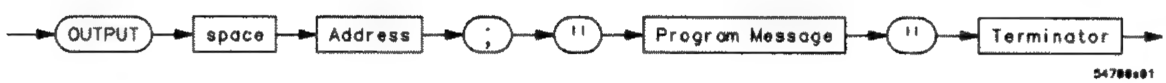
Programming Syntax

Computers acting as controllers communicate with the oscilloscope by sending and receiving program messages over a remote interface. Program messages are placed on the bus using an input or output command and passing the device address, instruction, and terminator. Passing the device address ensures that the instruction is sent to the correct interface and instrument. Instructions for programming the oscilloscope normally appear as ASCII character strings embedded inside the output statement of a "host" language available on your controller. Responses from the oscilloscope are read with the input statements of the host language.

HP BASIC Output Statement

HP 9000 Series 200/300 BASIC uses the OUTPUT statement for sending commands and queries to the oscilloscope.

Figure 5-1

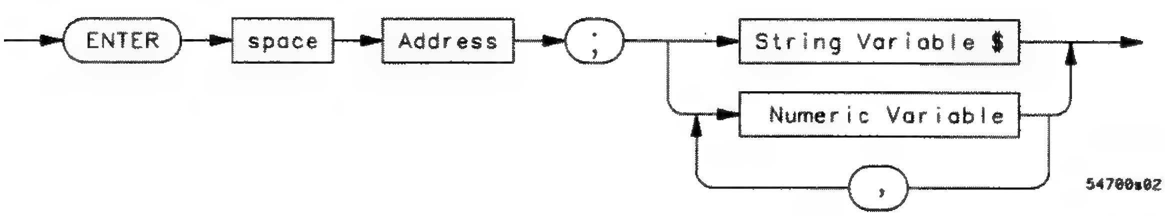


HP Basic Syntax for Sending Program Messages

HP BASIC Enter Statement

After a query is sent, the response is usually read using the HP BASIC ENTER statement. The ENTER statement passes the value across the bus to the controller and places it in the designated variable.

Figure 5-2



HP Basic Syntax for Receiving Responses

Device Address

The examples in this manual assume the oscilloscope is at device address 707. In HP BASIC, the address is specified after the keyword OUTPUT or ENTER. In actual programs, the number you use varies according to how you have configured the bus for your application.

Instructions

Instructions can be sent to the oscilloscope in either the long form (complete spelling) or the short form (abbreviated spelling). Upper-case and lower-case letters may be mixed freely. When receiving responses from the instrument, upper-case letters are used exclusively. The use of the long form or short form in a response depends on the setting you last specified with the SYSTEM:LONGFORM command.

Instructions are composed of two main parts:

- The header, which specifies the command or query to be sent.
- The program data, which provide additional information needed to clarify the meaning of the instruction.

Instruction Header

Colons

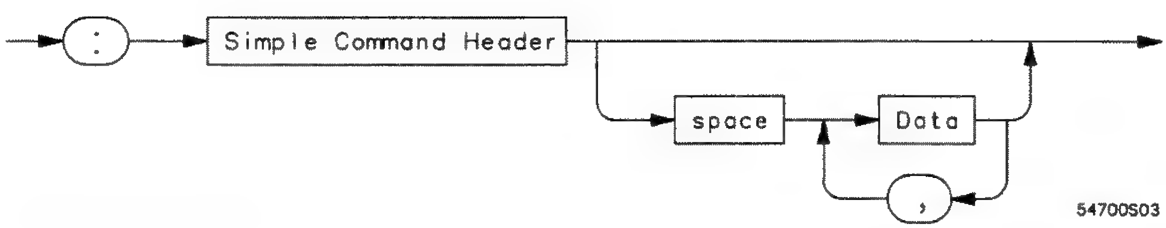
The instruction header is one or more mnemonics separated by colons (:) that represent the operation to be performed by the instrument. There are three types of headers:

- Simple Command headers.
- Compound Command headers.
- Common Command headers.

Simple Command Headers

Simple command headers contain a single mnemonic. AUTOSCALE and DIGITIZE are examples of simple command headers typically used in this oscilloscope.

Figure 5-3

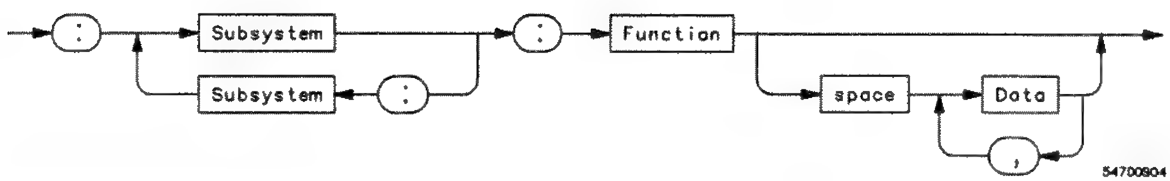


Simple Command Syntax

Compound Command Headers

Compound command headers are a combination of two or more program mnemonics. The first mnemonic selects the subsystem, and the last mnemonic selects the function within the sybsystem. Additional mnemonics may appear between the subsystem mnemonic and the function mnemonic when there are additional levels within the subsystem that must be transversed. The mnemonics within the compound header are separated by colons. An example of a compound header is :SYSTEM:LONGFORM.

Figure 5-4

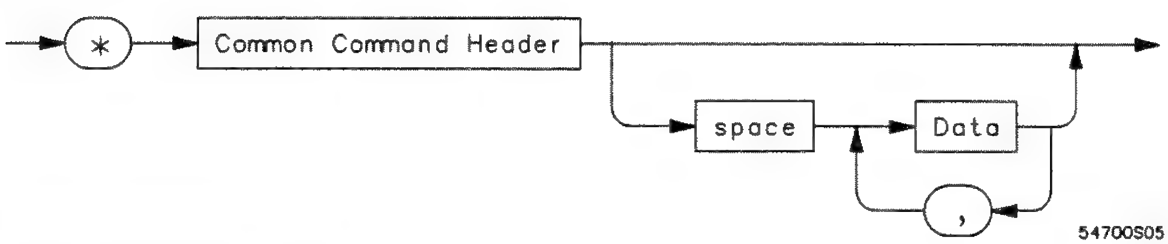


Compound Command Syntax

Common Command Headers

Common command headers control IEEE 488.2 functions within the instrument such as clearing the status (*CLS). No space or separator is allowed between the asterisk (*) and the command header.

Figure 5-5



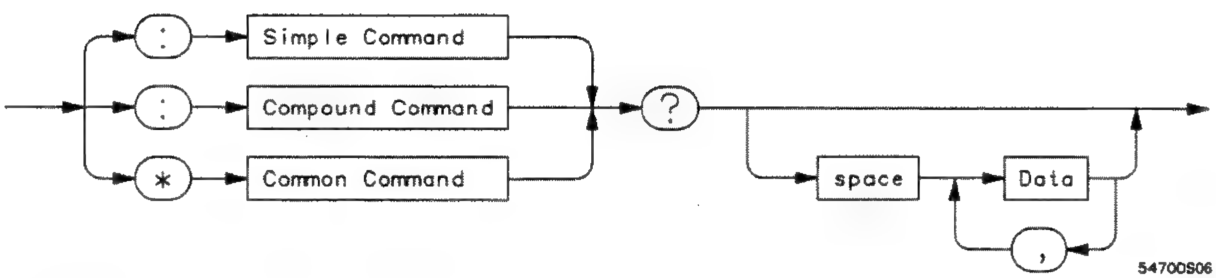
Common Command Syntax

Queries

Headers immediately followed by a question mark (?) are queries. After receiving a query, the instrument interrogates the required function and places the answer in its output queue. The answer remains in the output queue until it is read or another command is issued. When read, the answer is transmitted across the bus to the designated listener (typically a controller). For example, the query :TIMEBASE:RANGE? places the current time base setting in the output queue.

Sending another command or query before reading the result of a query causes the output queue to be cleared and the current response to be lost. This also generates an error in the error queue.

Figure 5-6



Query Syntax

Queries can be used to find out how the instrument is currently configured. They are also used to get results of measurements made by the oscilloscope, with the query actually activating the measurement.

Program Data

Program data are used to clarify the meaning of a command or query. They provide necessary information such as whether a function should be on or off, which waveform is to be displayed, and more.

Spaces and Commas

A space separates the header from the data. When there is more than one data parameter, the data parameters are separated by commas (,).

Character Program Data

Character program data is used to convey parameter information as alpha or alphanumeric strings. The available mnemonics for program data are listed with the individual commands in this manual.

Numeric Program Data

With numeric program data, you have the option of using exponential notation or using suffix multipliers to indicate a numeric value. The following numbers are all equal:

28 = 0.28E2 = 280E-1 = 28000m = 0.028K = 28E-3K

When a syntax definition specifies that a number is an integer, that means the number should be whole without any fractional part or decimal point.

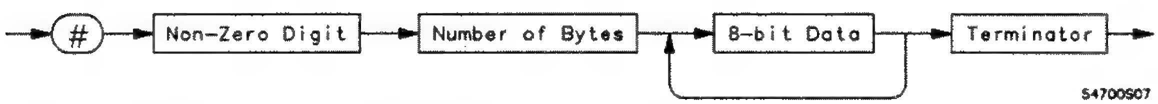
Embedded Strings

Embedded strings contain groups of alphanumeric characters that are treated as a unit of data by the oscilloscope. For example, the line of text written to the advisory line of the oscilloscope with the :SYSTEM:DSP command. Embedded strings may be delimited with either single (') or double (") quotes. These strings are case-sensitive and spaces act as legal characters just like any other character.

Block Data

Definite-length block response data (block data) allows any type of device-dependent data to be transmitted over the bus as a series of 8-bit binary data bytes. This is particularly useful for sending large quantities of data or 8-bit extended ASCII codes. The syntax is a pound sign (#) followed by a non-zero digit representing the number of digits in the decimal integer. After the non-zero digit is a decimal integer that states the number of 8-bit data bytes being sent. This is followed by the actual data.

Figure 5-7



Block Data Syntax

For example, for transmitting 500 bytes of data, the syntax would be:

#3500<500 bytes of data><terminator>

The "3" states the number of digits that follow, and the "500" states the number of bytes to be transmitted.

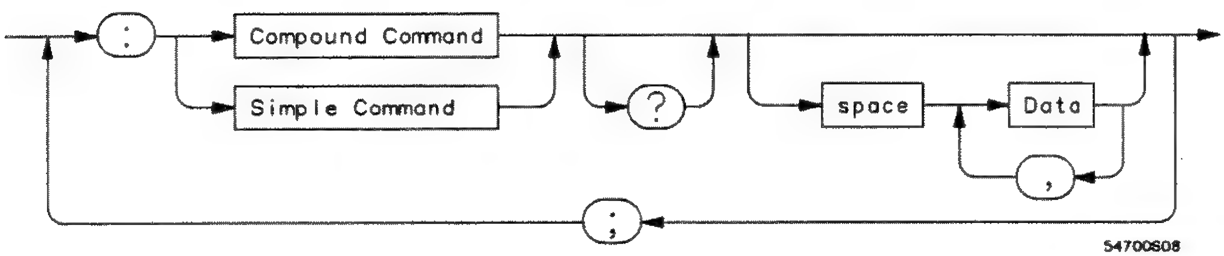
Multiple Subsystems

Semi-Colon

You can send multiple instructions (commands and queries) for different subsystems on the same line by separating each instruction with a semi-colon (;). The colon following the semi-colon enables you to enter a new subsystem (for example :CHANNEL1:RANGE 0.4;:TIMEBASE:RANGE 1).

Multiple instructions may be any combination of compound and simple commands.

Figure 5-8



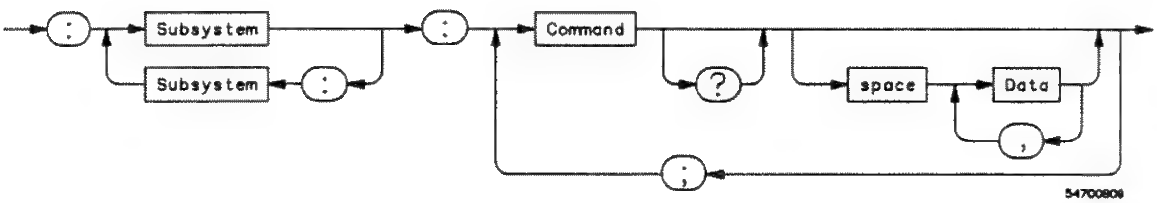
Selecting Multiple Subsystems

Multiple Functions within a Subsystem

Semi-Colon

To execute more than one function within the same subsystem, separate each function with a semi-colon (;). For example :SYSTEM:LONGFORM ON;HEADER ON turns the long form on and the headers on.

Figure 5-9



Selecting Multiple Functions within a Subsystem

Common Commands within a Subsystem

Common commands can be received and processed by the oscilloscope whether they are sent over the bus as separate program messages or within other program messages. If a subsystem has been selected and a common command is received by the oscilloscope, the instrument remains in the selected subsystem. For example, if the program message

```
":ACQUIRE:TYPE AVERAGE;*CLS;COUNT 1024"
```

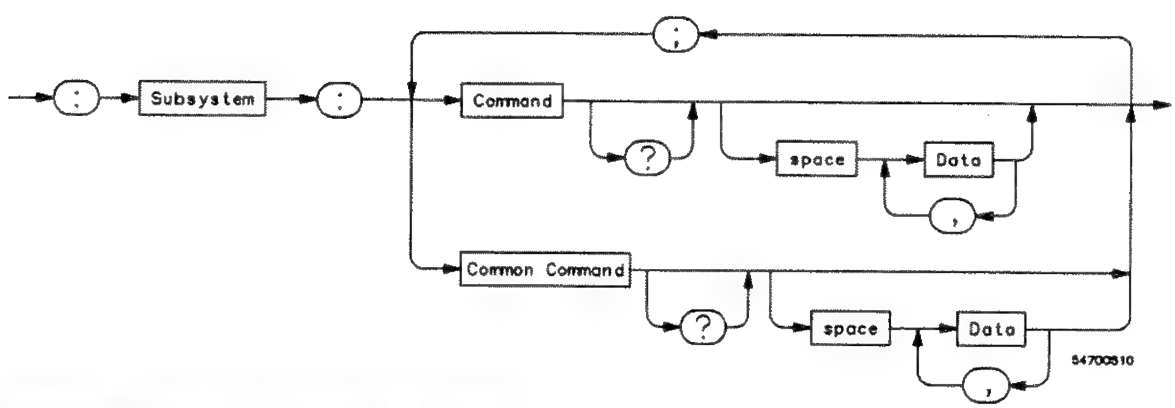
is received by the oscilloscope, the oscilloscope sets the acquire type and count, then clears the status information without leaving the selected subsystem.

If some other type of command is received within a program message, you must reenter the original subsystem after the command. For example, the program message

```
"ACQUIRE:TYPE AVERAGE;:AUTOSCALE;ACQUIRE:COUNT 1024"
```

sets the acquire type, completes the autoscale operation, then sets the acquire count. In this example, :ACQUIRE must be sent again after the AUTOSCALE command in order to reenter the acquire subsystem and set count.

Figure 5-10

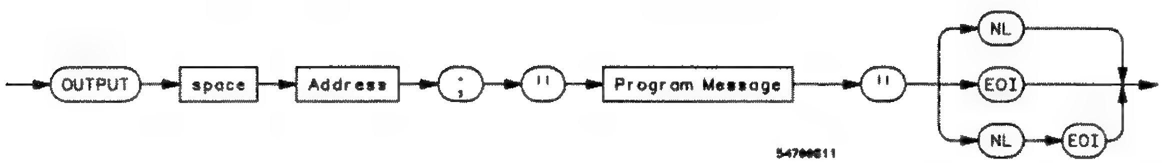


Selecting Common Commands within a Subsystem

Instruction Terminator

The instructions within the program message are executed after the instruction terminator is received. The terminator may be either a New Line (NL) character, and End-Or-Identify (EOI) asserted, or a combination of the two. All three ways are equivalent. Asserting the EOI sets the EOI control line low on the last byte of the data message. The NL character is an ASCII linefeed (decimal 10).

Figure 5-11



Instruction Terminator

6

Programming Conventions

Programming Conventions

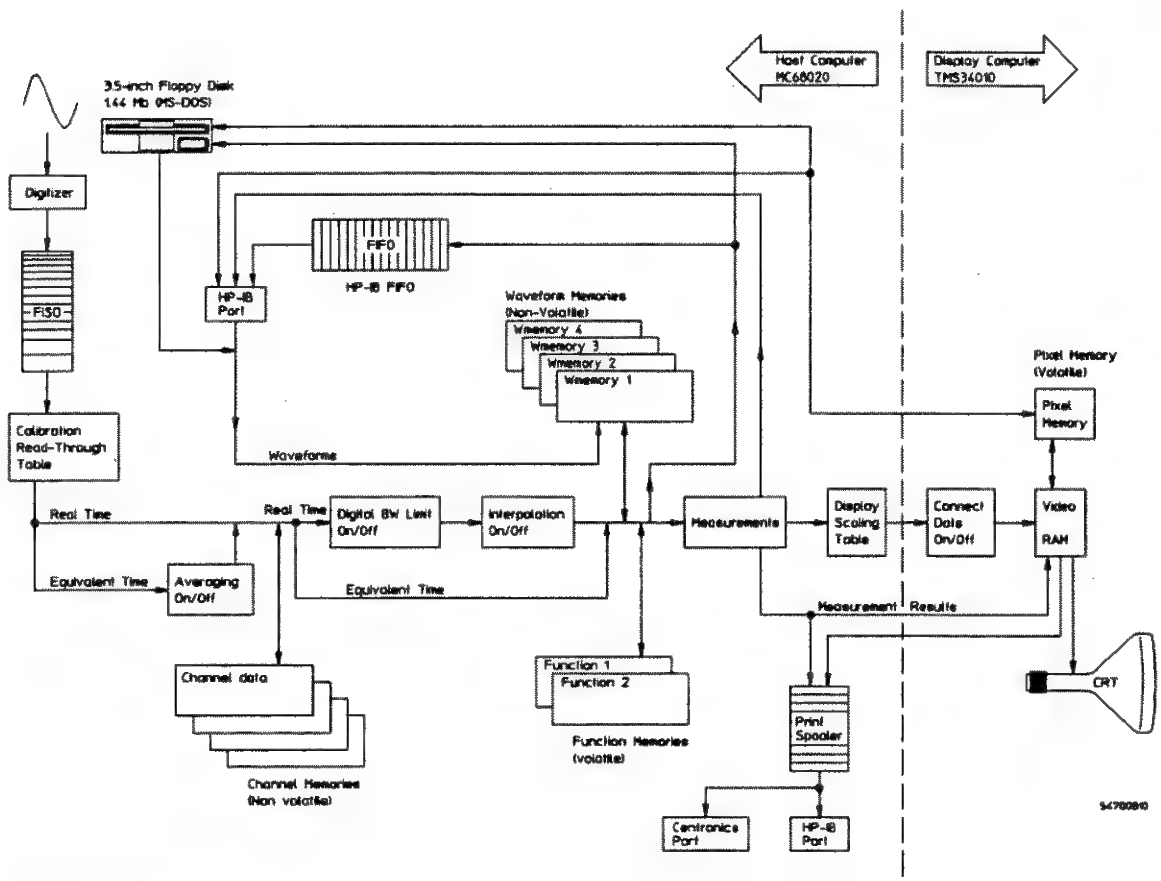
This chapter covers conventions used in programming the oscilloscope, as well as conventions used throughout this manual. A block diagram and description of data flow is included for understanding oscilloscope operations. A detailed description of the command tree and command tree traversal is also included in this chapter.

Data Flow

The data flow gives you an idea of where the measurements are made on the acquired data, and when the post-signal processing is applied to the data.

Figure 6-1 is a data flow diagram of the oscilloscope. The diagram is laid out serially for a visual perception of how the data is affected by the oscilloscope.

Figure 6-1



Oscilloscope Data Flow

The digitizer samples the applied signal and converts it to a digital signal. The FISO holds the data until the system bus is ready for the data. The output of the FISO is raw data, and it is used as an address to the calibration read-through table (cal table).

The cal table automatically applies the calibration factors to the raw data, so that the output of the cal table is calibrated data.

In the real-time sampling mode, the calibrated data is stored in the channel memories before any of the postprocessing is performed. Postprocessing includes turning on or off the digital bandwidth limit filter or the interpolator, calculating functions, storing data to the waveform memories, transferring data over the HP-IB bus, or transferring data to and from the disk. Notice that the measurements are performed on the real-time data after it has gone through postprocessing.

Therefore, you can make measurements on the data, and you can turn on or off digital bandwidth limit or interpolation without having to reacquire the data. This is important because the real-time sampling mode is primarily used on events that happen either once or infrequently, and reacquiring the data may not be one of your options. Also, turning on interpolation usually improves the repeatability of your measurements.

The equivalent-time sampling mode is slightly different. Notice that averaging is turned on or off before the data is stored in the channel memories. That means once the data is acquired, if you need to turn averaging on or off before making any measurements, you must reacquire the data. However, because the equivalent-time sampling mode is primarily used on repetitive signals, you should be able to reacquire the data.

Also, you may notice that postprocessing the data in the equivalent-time signal path includes calculating functions, storing data to the waveform memories, transferring data over the HP-IB bus, or transferring data to and from the disk.

After the measurements are performed, the data is sent through the display portion of the oscilloscope. Notice that connected dots is a display feature, and that it has no influence on the measurement results. The pixel memory is also part of the video RAM, which is past the point where the measurements are performed on the data. Therefore, you cannot make measurements on data in the pixel memory. But, you can make measurements on data stored to the waveform memories.

Truncation Rule

The following truncation rule is used to produce the short form (abbreviated spelling) for the mnemonics used in the programming headers and alpha arguments.

Truncation Rule

The mnemonic is the first four characters of the keyword, unless the fourth character is a vowel. Then the mnemonic is the first three characters of the keyword.

If the length of the keyword is four characters or less, this rule does not apply and the short form is the same as the long form.

The following table shows how the truncation rule is applied to various commands.

Table 6-1 **Mnemonic Truncation**

Long Form	Short Form	How The Rule is Applied
RANGE	RANG	Short form is the first four characters of the keyword.
PATTERN	PATT	Short form is the first four characters of the keyword.
TIME	TIME	Short form is the same as the long form.
DELAY	DEL	Fourth character is a vowel, short form is the first three characters.

The Command Tree

The command tree in figure 6-2 shows all of the commands in this oscilloscope and the relationship of the commands to each other. The IEEE 488.2 common commands are not listed as part of the command tree since they do not affect the position of the parser within the tree.

When a program message terminator (<NL>, linefeed - ASCII decimal 10) or a leading colon (:) is sent to the instrument, the parser is set to the "root" of the command tree.

Command Types

The commands in this instrument can be placed into three types: Common commands, root level commands, and subsystem commands.

- Common commands are commands defined by IEEE 488.2 and control some functions that are common to all IEEE 488.2 instruments. These commands are independent of the tree and do not affect the position of the parser within the tree. *RST is an example of a common command.
- Root level commands control many of the basic functions of the instrument. These commands reside at the root of the command tree. They are always parsable if they occur at the beginning of a program message, or are preceded by a colon. Unlike common commands, root level commands place the parser back at the root of the command tree. AUTOSCALE is an example of a root level command.
- Subsystem commands are grouped together under a common node of the command tree, such as the TIMEBASE commands. Only one subsystem may be selected at a given time. When the instrument is initially turned on, the command parser is set to the root of the command tree and no subsystem is selected.

Tree Traversal Rules

Command headers are created by traversing down the command tree. A legal command header from the command tree in figure 6-2 would be :TIMEBASE:RANGE. This is referred to as a compound header. A compound header is a header made up of two or more mnemonics separated by colons. The mnemonic created contains no spaces. The following rules apply to traversing the tree.

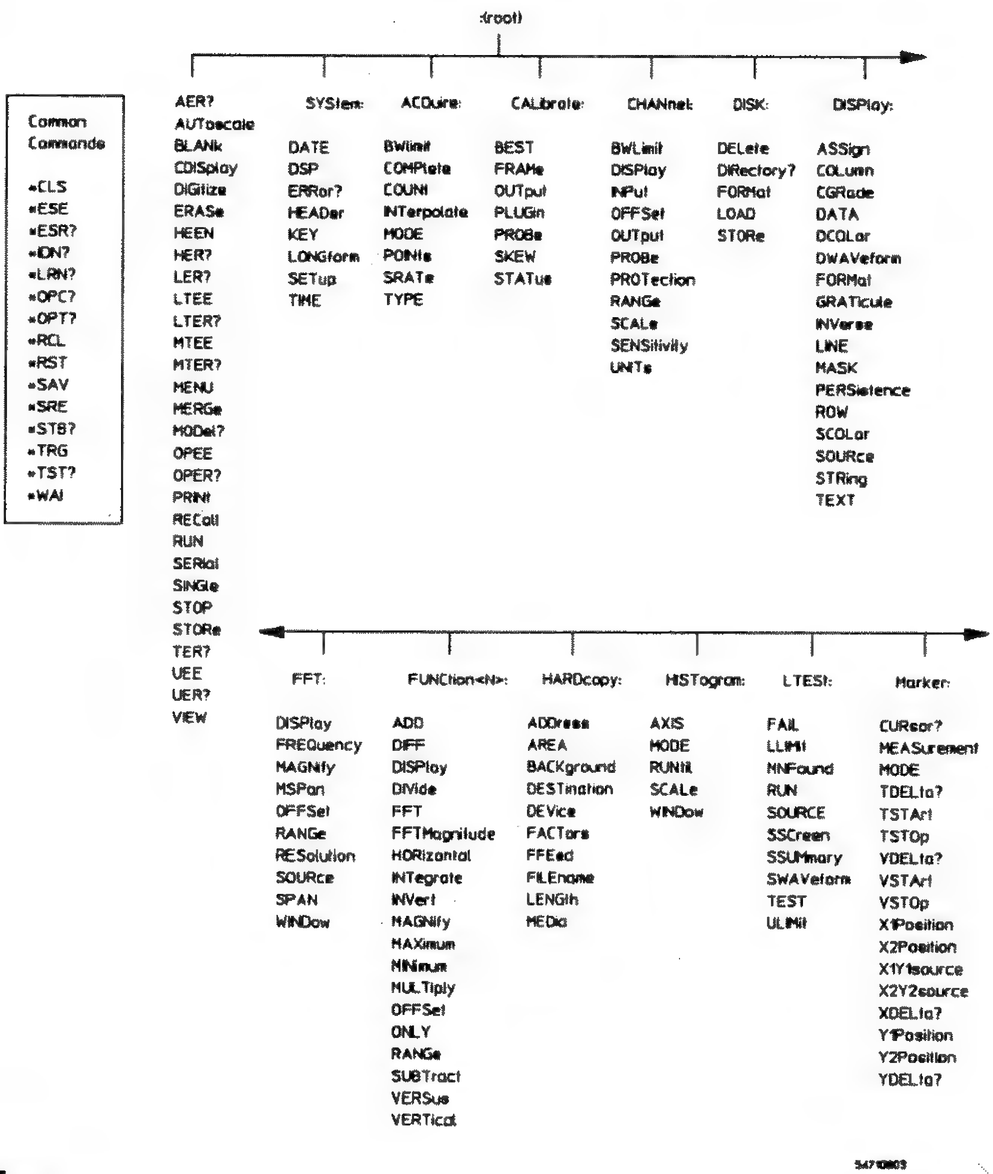
Tree Traversal Rules

A leading colon or a program message terminator (either an <NL> or EOI true on the last byte) places the parser at the root of the command tree. A leading colon is a colon that is the first character of a program header.

Executing a subsystem command places you in that subsystem until a leading colon or a program message terminator is found.

In the command tree of figure 6-2, use the last mnemonic in the compound header as a reference point (for example, RANGE). Then find the last colon above that mnemonic (TIMEBASE:). That is the point where the parser resides. Any command below this point can be sent within the current program message without sending the mnemonics which appear above them (for example, REFERENCE).

Figure 6-2



Command Tree

Figure 6-2 (continued)

MEASure:	MTESt:	Timebase:	TRIGger:	TRIGger<N>:	PMEMory:	WMEMory<N>:	WAVeform:
DEFine	AMASk	DELay	DEVents	BWLimit	ADD	DISPlay	BANDpass?
DELTatime	COUNt	POSition	DTIMe	PROBe	CLEAR	SAVe	BYTeorder
DUTycycle	MASK	RANGe	EDGe		DISPlay	XOFFset	COMPlete?
FALLtime	POLYgon	REFerence	GLITch		ERASe	XRANge	COUNt?
FFT	RUMode	SCALe	HOLDoff		MERGe	YOFFset	COUPling?
FREQuency	SCALe	VIEW	HYSTeresis			YRANge	DATA
HISTogram	SSCReen	WINDow	LEVel				FORMat
NWIDth	SSUMmary		MODE				POINts?
OVERshoot	SWAVeform		PATTern				PREamble
PERiod	TEST		SLOPe				SOURce
PREShoot			SOURce				TYPE?
PWIDth			STATe				VIEW
RESults?			SWEep				XDISplay?
RISetime							XINCrement?
SCRatch							XORigin?
SENDvalid							XRANge?
SOURce							XREFerence?
STATistics							XUNits?
TEDGe							YDISplay?
TMAX?							YINCrement?
TVOLt?							YORigin?
VAMPlitude							YRANge?
VAVerage							YREFerence?
VBASe							YORigin?
VLOWer							YREFerence?
VMAX							YUNits?
VMIN							
VPP							
VRMS							
VTIMe							
VTOP							
VUPper							

54710B04

Command Tree (continued)

Tree Traversal Examples

The OUTPUT statements in the following examples are written using HP BASIC 5.0 on an HP 9000 Series 200/300 controller. The quoted string is placed on the bus, followed by a carriage return and linefeed (CRLF).

Example 1

```
OUTPUT 707;":CHANNEL1:RANGE 0.5;OFFSET 0"
```

In the previous example, the colon between CHANNEL1 and RANGE is necessary because CHANNEL1:RANGE is a compound command. The semicolon between the RANGE command and the OFFSET command is required to separate the two functions. The OFFSET command does not need CHANNEL1 preceding it since the CHANNEL1:RANGE command sets the parser to the CHANNEL1 node in the tree.

Example 2

```
OUTPUT 707;":TIMEBASE:REFERENCE CENTER;POSITION 0.00001"
```

or

```
OUTPUT 707;":TIMEBASE:REFERENCE CENTER"
OUTPUT 707;":TIMEBASE:POSITION 0.00001"
```

In the first line of example 2, the "subsystem selector" is implied for the POSITION command in the compound command.

A second way to send these commands is shown in the second part of the example. Since the program message terminator places the parser back at the root of the command tree, TIMEBASE must be re-selected to re-enter the TIMEBASE node before sending the POSITION command.

Example 3

OUTPUT 707;":TIMEBASE:REFERENCE CENTER;:CHANNEL1:OFFSET 0"

In example 3, the leading colon before CHANNEL1 tells the parser to go back to the root of the command tree. The parser can then recognize the CHANNEL1:OFFSET command and enter the correct node.

Infinity Representation

The representation for infinity for this oscilloscope is 9.99999E+37. This is also the value returned when a measurement cannot be made.

Sequential and Overlapped Commands

IEEE 488.2 makes a distinction between sequential and overlapped commands. Sequential commands finish their task before the execution of the next command starts. Overlapped commands run concurrently. Commands following an overlapped command may be started before the overlapped command is completed.

Response Generation

As defined by IEEE 488.2, query responses may be buffered for the following reasons:

- When the query is parsed by the instrument.
- When the controller addresses the instrument to talk so that it may read the response.

This oscilloscope buffers responses to a query when the query is parsed.

EOI

The EOI bus control line follows the IEEE 488.2 standard without exception.

7

Example Programs

Example Programs

The programs listed in this chapter are the same as those on the disks provided with this programmer's reference. The disks are provided in both LIF and DOS formats. The disks contain some interactive files, i.e., additional files are created while running the programs. To preserve the original quality of the example programs disks, make a copy of the originals and use the copy for running the programs.

Digitize Example Program

```
10    ! "DIG_7XX.ibw"
20    !
30    !         Copyright:  (c) 1993, Hewlett-Packard Co.  All rights reserved.
40    !       Contributor:  Colorado Springs Division
41    !           Product:  Example Program
42    !
43    ! $Revision:  3.0
44    ! $Date$      6.9.93
45    ! $Author$    Ed Mierzejewski
46    !
47    ! Description:  DIG_7XX.ibw autoscales to get a waveform on screen and
48    !               digitizes the waveform.  Then the operator can reposition
49    !               before transfering the data to the computor.  Then the
50    !               computor will draw the waveform as repositioned on the
51    !               computor screen.  It also save the data to a record and
52    !               recalls that data before drawing it.
53    !
54    ! Main Routine:  Begin_main.
55    ! Sub routines:  none.
56    ! Sub programs:  Get_waveform, Graph, Initscope, Readme, Readme2,
57    !                Retrieve_wave, Save_waveform.
59    ! Functions:     none.
60    ! Variable List: Preamble & Waveform, @Path, & @Scope
61    !
62    !      Preamble = Real array for the first 15 parameters of the
63    !                 preamble, they are numerics and the remaining 3
64    !                 parameters are alphas and are not used.
66    !      Waveform = Integer array to store the wavefrom data.
67    !      @Path = the path for saving/recalling data to/from media.
68    !      @Scope = The scope's complete HPIB address.
70    !
71    REAL Preamble(1:15)
130   INTEGER Waveform(1:32000)
140 Begin_main:  !
185   CALL Readme
186   CALL Initscope(@Scope)
190   CALL Get_waveform(@Scope,Waveform(*),Preamble(*))
220   CALL Save_waveform(@Path,Waveform(*),Preamble(*))
240   CALL Readme2
260   CALL Retrieve_wave(@Path,Waveform(*),Preamble(*))
290   CALL Graph(Waveform(*),Preamble(*))
```

```
291   PRINT TABXY(15,30);"Program has Ended."
292   LOCAL 707
300 End_main:  !
310   END
320 Begin_subs:!
321   !
330   SUB Readme
340   ! Description:  Readme prints program explaination to the computor
341   !               screen.
342   ! Parameters: none.
343   !
350     CLEAR SCREEN
360     PRINT "DIG_7XX.ibw does the following tasks:"
370     PRINT
380     PRINT "              a.  initialize interface and scope"
390     PRINT "              b.  digitize and acquire data"
400     PRINT "              c.  store data to disk"
410     PRINT "              d.  retrieve data from disk"
420     PRINT "              e.  draw signal on computer"
430     PRINT
440     PRINT "Assumed system configuration is:"
450     PRINT
460     PRINT "    HP-IB address = 7"
470     PRINT "    Scope address = 7"
480     PRINT "    signal attached to channel 1"
490     PRINT
500     PRINT "If the addresses are not correct, change the ASSIGN "
510     PRINT "statements in sub program 'Initscope'."
520     PRINT
530     PRINT "Press Continue when ready to start"
540     PAUSE
550     CLEAR SCREEN
560   SUBEND
570     !
580   SUB Readme2
590     !
600     !  Description:  Readme2 is user information and status.
610     !
611     !  Parameters: none.
620     !
621     CLEAR SCREEN
623     PRINT
630     PRINT "The waveform data and preamble information have now been"
640     PRINT "read from the scope and stored in the computer's disk."
641     PRINT
```

```
642     PRINT "When you press continue that information will be retrieved"
650     PRINT "from the disk, and plotted to the computor screen."
680     PRINT
690     PRINT "Press CONTINUE to continue."
700     PAUSE
710     CLEAR SCREEN
720   SUBEND
730     !
740   SUB Initscope(@Scope)
750     !
760     ! Description:  Initscope assigns the path to the scope, initializes
761     !               the scope, autoscales, and sets up the acquisiton
762     !               parameters.
763     ! Parameters:
764     !     Passed: @Scope = the HPIB address of the scope.
765     !   Internal: @Isc = interface select code of the HPIB interface.
766     ! Modified Variables: @Scope and @Isc
773     !
774     CLEAR SCREEN
780     PRINT "INITIALIZE"
781 Assign_paths:    !
790     ASSIGN @Isc TO 7                    ! Interface Select Code = 7
800     ASSIGN @Scope TO 707                ! scope address
801 Init_sys:    !
810     CLEAR @Isc                          ! clear HP-IB interface
820     OUTPUT @Scope;"*RST;*CLS"           ! set scope to default config
830     OUTPUT @Scope;":AUToscale"
840     OUTPUT @Scope;":SYStem:HEADer OFF"
850 Acq_setup:    !
890     OUTPUT @Scope;":ACQuire:COMPlete 100;POINts 500"
910     OUTPUT @Scope;":WAVeform:FORMat BYTE;SOURce CHANnel1"
911     !
920     ! Normally WORD data would be recommended because it allows better
930     ! use of the full resolution of the scope, especially in ET MODE.
940     ! Byte data is shown because HPBasic doesn't recognize signed bytes
950     ! and requires a conversion.  FNBcon will do the conversion.
960     !
980     CLEAR SCREEN
990   SUBEND
1000    !
1010  SUB Get_waveform(@Scope,INTEGER Waveform(*),REAL Preamble(*))
1020    !
1021    ! Description:  Get_waveform digitizes the autoscaled waveform,
1022    !               gets wavefrom data and preamble after the operator
1023    !               adjusts the display to show the data as desired.
```

```
1050    !
1051    !                 There are 2 forms of digitize:  1.  'with parameters'
1052    !                 will digitize the specified channel/function, screen
1053    !                 is blanked, then place the data in associated channel/
1054    !                 function memory.  2.  'without parameters' digitizes
1055    !                 all 'on' channels/functions and places data in the
1056    !                 associated channel/function memory, and leaves them on
1057    !                 but stopped.
1058    !
1059    !                 Both digitizes are here and on adjacent lines.  One of
1060    !                 the lines must be commented out or only the last one
1061    !                 will be used.
1063    !
1064    ! Parameters:
1065    !   Passed:  @Scope, Waveform, Preamble
1066    !   Internal:  Digits = this is the length of the data header.
1067    !              Length = the number of bytes of data from the scope.
1068    !              End$   = enpties output buffer of linefeed.
1069    !              One_char$ = used to find the '#' character.
1070    !
1072    ! Modified Variables:  Waveform, Preamble, Digits, Length, End$, and
1073    !                      One_char$.
1074    !
1075    CLEAR SCREEN
1076    PRINT "Get_waveform"
1077 !  OUTPUT @Scope;":DIGitize CHAN1"
1080    OUTPUT @Scope;":DIGitize"
1090 User_sets_disp:   !
1150    LOCAL 707
1160    PRINT "Adjust Display as you want it.  Press continue when ready."
1170    PAUSE
1171 Read_data:   !
1180    OUTPUT @Scope;":WAVeform:DATA?"
1190    ENTER @Scope USING "#,1A,";One_char$
1200    IF One_char$="#" THEN
1220      ENTER @Scope USING "#,1D";Digits
1230      ENTER @Scope USING "#,"&VAL$(Digits)&"D";Length
1231      CLEAR SCREEN
1233      PRINT
1240      PRINT "reading ";Length;" bytes from scope"
1250       !
1260       !Redimention the array for the waveform data.  After data is
1270       !read in, one extra byte read to clear the line feed (10)
1280       !attached to the end of the scope's output buffer.
1290       !
```

```
1300       REDIM Waveform(1:Length)
1310       ENTER @Scope USING "#,B";Waveform(*)
1320       ENTER @Scope USING "-K,B";End$
1330       OUTPUT @Scope;":WAVEFORM:PREAMBLE?"
1340       ENTER @Scope;Preamble(*)
1370     ELSE
1380       PRINT "BAD DATA"
1390     END IF
1400   SUBEND
1410   !
1420   SUB Save_waveform(@Path,INTEGER Waveform(*),REAL Preamble(*))
1430   !
1431   ! Description:  Save_waveform sends acquired data and preamble to the
1432   !               computor's disk.  It is stored in 'WAVESAMPLE'.  If
1433   !               'WAVESAMPLE' already exist, it will be purged then a
1434   !               new one created.
1435   !
1436   ! Parameters:
1437   !    Passed:  @Path, Waveform, Preamble
1438   !    Internal:  none
1439   !
1440   ! Modified Variables: none
1441   ! Sub programs: Ertrap
1442   !
1490     ON ERROR CALL Ertrap
1500     CREATE BDAT "WAVESAMPLE",1,4080
1510     ASSIGN @Path TO "WAVESAMPLE"
1520     OUTPUT @Path;Waveform(*),Preamble(*)
1530   SUBEND
1540   !
1550   SUB Retrieve_wave(@Path,INTEGER Waveform(*),REAL Preamble(*))
1551   !
1552   ! Description: Retrieve_wave reads data and preamble stored in
1560   !              'WAVESAMPLE'.
1561   ! Parameters:
1562   !     Passed:  @Path, Waveform, Preamble
1563   !    Internal:  Con = indexing variable
1564   ! Functions:  FNBcon = converts from signed bytes.
1565   !
1590     ASSIGN @Path TO "WAVESAMPLE"
1600     ENTER @Path;Waveform(*),Preamble(*)
1610     FOR Con=1 TO Preamble(3)
1620       Waveform(Con)=FNBcon(Waveform(Con))
1630     NEXT Con
1640   SUBEND
```

```
1650  !
1660  SUB Graph(INTEGER Waveform(*),REAL Preamble(*))
1670  !
1680  ! Description:  Graph takes the converted data and plots it on screen.
1690  !               It uses the 'Y Display Range' to show the data as seen
1700  !               on screen vertically, and 'X Display Range' to show
1710  !               as seen horizontally (pre(14 and 12 respectively).
1720  !
1730  ! Parameters: Waveform(*) = array of data values.  Enters as q levels
1740  !                           leaves as voltages.
1750  !              Preamble(*) = the preamble for the data.
1760  !
1770  !  Internal: Vrange = preamble(14), y-axis duration of waveform displayed.
1780  !            Srange = preamble(12), x-axis duration of waveform displayed.
1790  !            Offset = preamble(15), center of screen vertically.
1800  !            vmin   = lower limit vertically.
1810  !            vmax   = upper limit vertically.
1820  !            hmin   = lower limit horizontally (preamble(13)).
1830  !            hmax   = upper limit horizontally.
1840  !            Hdata(*) = Horizontal values in proper units.
1850  !            Vdata(*) = Vertical values in proper units.
1851  !            I = indexing variable.
1860  !
1870  ! Modified variables: Hdata(*), Vdata(*), and I
1880  !
1890  ! Subprogram calls:V_convert and H_convert.
1900  !
1910    ALLOCATE REAL Hdata(1:Preamble(3))
1920    ALLOCATE REAL Vdata(1:Preamble(3))
1930    CALL V_convert(Waveform(*),Preamble(*),Vdata(*))
1940    CALL H_convert(Hdata(*),Preamble(*))
1950    Vrange=Preamble(14)
1960    Srange=Preamble(12)
1970    Offset=Preamble(15)
1980    Vmin=Offset-Vrange/2
1990    Vmax=Vrange/2+Offset
2000    Hmin=Preamble(13)
2010    Hmax=Hmin+Srange
2020    GCLEAR                              !initialize graphics
2030    CLEAR SCREEN
2040    GINIT
2050    GRAPHICS ON
2060    VIEWPORT 0,130,35,100
2070    WINDOW Hmin,Hmax,Vmin,Vmax
2080    FRAME
```

```
2090    PEN 4
2100    MOVE Hdata(1),Vdata(1)
2110    FOR I=1 TO Preamble(3)              !plot data points
2120      DRAW Hdata(I),Vdata(I)
2130    NEXT I
2140    PRINT TABXY(0,18),"Vertical=";Vrange/8;" V/div";TAB(50),"Offset =
";Offset;"V"
2150    PRINT TABXY(0,19),"Time=";Srange/10;" s/div"
2160    DEALLOCATE Hdata(*)
2170    DEALLOCATE Vdata(*)
2180  SUBEND
2190  !
2200  DEF FNBcon(INTEGER B)
2201  !
2202  ! Description:  FNBcon takes the signed byte value from the scope and
2203  !               converts it to a positive integer of the proper value.
2204  ! Parameters:
2205  !    Passed: B
2206  !  Internal: Orparam = value to OR with the passed value, B, when the
2207  !                      MSB is set.
2208  ! Modified Variables:  B
2209  !
2260    Orparam=-256
2270    IF BIT(B,7) THEN B=BINIOR(Orparam,B)
2280    RETURN B
2290  FNEND
2300  !
2310  SUB Ertrap
2311  !
2312  ! Description:  Ertrap is called by an error interupt.  It checks for
2313  !               error #54 which will occur when there is a duplicate
2314  !               file name.  The existing file will be purged.
2315  ! Parameters: none
2316  !
2350    IF ERRN=54 THEN PURGE "WAVESAMPLE"
2360    OFF ERROR
2370  SUBEND
2380  !
2390  SUB V_convert(INTEGER Wav(*),REAL Pre(*),Vdata(*))
2400  !
2410  ! Description:  V_convert takes the data from the scope and converts it
2420  !               into voltage values using the equation from the manual.
2430  !
2440  ! Parameters: Wav(*) = array of data values.  Enters as q levels
2450  !                      leaves as voltages.
```

```
2460 !              Pre(*)    = the preamble for the data.
2470 !              Vdata (*) = array of vertical values, volts.
2480 !
2490 !  Internal: yref = pre(10), level associated with y origin.
2500 !            yinc = pre(8), duration between y-axis levels.
2510 !            yorg = pre(9), y-axis value at level zero.
2511 !               C = indexing variable.
2520 !
2530 ! Modified variables: Vdata(*)
2540 !
2550   Yref=Pre(10)
2560   Yinc=Pre(8)
2570   Yorg=Pre(9)
2580   FOR C=1 TO Pre(3)
2590     Vdata(C)=(Wav(C)-Yref)*Yinc+Yorg
2600   NEXT C
2610 SUBEND
2620 !
2630 SUB H_convert(Hdata(*),Pre(*))
2640 !
2650 ! Description:  H_convert creates horizontal axis values using the
2660 !               equation from the manual.
2670 !
2680 ! Parameters: Hdata(*) = Horizontal values.
2690 !             Pre(*)   = the preamble for the data.
2700 !
2710 !  Internal: xref = pre(7), data point associated with the x origin.
2720 !            xinc = pre(5), duration between x-axis data points.
2730 !            xorg = pre(6), x-axis value of first point in record.
2740 !
2750 ! Modified variables: Hdata(*)
2760 !
2770   Xref=Pre(7)
2780   Xinc=Pre(5)
2790   Xorg=Pre(6)
2800   FOR C=1 TO Pre(3)
2810     Hdata(C)=((C-1)-Xref)*Xinc+Xorg
2820   NEXT C
2830 SUBEND
```

Measurement Example Program

```
10 !  RE-SAVE "MEAS_7XX"  !Observing True Representation of signal.Rev. 1.18
20 !
30 !**********************************************************************!
40 !********************  Main Program **********************************!
50 !**********************************************************************!
60    Readme
70    Initscope(@Scope)
80    True_rep(@Scope)
90    Measure(@Scope)
100   Rt_4g(@Scope)
110   Measure(@Scope)
120   Rt_2g(@Scope)
130   Measure(@Scope)
140   PRINT "  The program has completed! "
150   BEEP
160          !*******************************************************************!
170   END    !*                       End of Main Program                       *!
180          !*******************************************************************!
190          !                                                                   !
200          !*******************************************************************!
210          !*                      Begin Sub Programs                         *!
220          !*******************************************************************!
230   SUB Readme
240     !*********************************************************************!
250     !*  This sub program writes user program information to the screen.  *!
260     !*********************************************************************!
270     CLEAR SCREEN
280     PRINT "This example program will setup and measure the cal."
290     PRINT "signal from the 54721A with different sampling rates"
300     PRINT "and statistics.  It shows the difference sampling rate"
310     PRINT "makes when measuring a fast pulse."
320     PRINT
330     PRINT "The program assumes that the system is configured such that:"
340     PRINT "   HP-IB interface is at address 7."
350     PRINT "   Scope is at address 7."
360     PRINT "   A 54721A is installed into slots 1 & 2."
370     PRINT
380     PRINT "If these addresses are incorrect, break program and set addresses"
390     PRINT "as needed in the ASSIGN statements."
400     PRINT
410     PRINT "PRESS continue to run program"
```

```
420     PAUSE
430     CLEAR SCREEN
440   SUBEND
450     !
460   SUB Initscope(@Scope)
470     !*****************************************************************!
480     !*  This sub program initializes the I/O and scope.             *!
490     !*****************************************************************!
500     ASSIGN @Scope TO 707
510     CLEAR @Scope                      !clear HP-IB interface
520     OUTPUT @Scope;"*cls"
530     OUTPUT @Scope;"*RST"              !reset scope to default config
540     OUTPUT @Scope;":SYSTEM:HEADER OFF"!turn off header
550     CLEAR SCREEN
560   SUBEND
570   SUB True_rep(@Scope)
580     !*****************************************************************!
590     !*  The first step of this demonstration is to show the true    *!
600     !*  signal in the equivalent time mode.                         *!
610     !*****************************************************************!
620     PRINT "  First see the true signal in the Equivalent Time mode."
630     PRINT
640     PRINT " !****************************************************!"
650     PRINT "  Connect the HP 54721A Calibrator Output to the Input  "
660     PRINT "  of the 54721A.                                        "
670     PRINT " !****************************************************!"
680     PRINT
690     PRINT "  Press continue when ready to continue.  "
700     PAUSE
710     CLEAR SCREEN
720     OUTPUT @Scope;":channel1:display on"
730         !********** Turn on the  calibrator signal on the 21 plug-in.
740     OUTPUT @Scope;":channel1:output on"
750     OUTPUT @Scope;":acquire:mode ETIME"
760     OUTPUT @Scope;":autoscale"
770     OUTPUT @Scope;":display:persistence infinite"
780         !  This completes the first setup.  FTE26
790     OUTPUT @Scope;":menu acquire"
800     PRINT "Waiting 5 seconds for Autoscale to complete and to "
810     PRINT "acquire a waveform."
820     WAIT 5
830     CLEAR SCREEN
840     PRINT "  The displayed waveform on the 54720A is the true "
850     PRINT "  representation of the signal."
860     PRINT
```

```
870    PRINT "  The HP 54720A is in equivelent time mode:"
880    PRINT "      Sampling rate is 500 MSa/S,"
890    PRINT "      Analog Bandwidth is 1.1 GHz"
900    PRINT
910    PRINT "  Press Continue to Continue"
920        !  Save the True Waveform to a Pixel memory for later
930        !  comparison.
940    PAUSE
950    CLEAR SCREEN
960    OUTPUT @Scope;":pmemory1:clear"
970    OUTPUT @Scope;":pmemory1:display on"
980    OUTPUT @Scope;":pmemory1:add"
990    BEEP
1000   INPUT "Do you want to leave the saved waveform on?  <Y or N>",Y$
1010   IF UPC$(Y$[1,1])="Y" THEN
1020   ELSE
1030     OUTPUT @Scope;":pmemory1:display off"
1040   END IF
1050 SUBEND
1060 !
1070 SUB Rt_4g(@Scope)
1080   !*******************************************************!
1090   !* This sub program will reaquire the 54721A calibration *!
1100   !* output in the real time mode with 4GSa/s.            *!
1110   !*******************************************************!
1120   CLEAR SCREEN
1130   PRINT "Setting up for 4GSa/sec."
1140   OUTPUT @Scope;":acquire:mode rtime"
1150   OUTPUT @Scope;":acquire:srate 4E+9"
1160   OUTPUT @Scope;":acquire:interpolate on"
1170   PRINT
1180   PRINT "  When ready to make some measurements press continue."
1190   PAUSE
1200   CLEAR SCREEN
1210 SUBEND
1220 !
1230 SUB Measure(@Scope)
1240   !**************************************************************!
1250   !* This sub program will make a +width and Vpp measurement. "  *!
1260   !* It will also report the mean and standard deviations.    "  *!
1270   !**************************************************************!
1280   CLEAR SCREEN
1290   REAL R(1:14),A(1:14)
1300   DIM Ex$(1:14)[1],M$[32]
1310   !
```

```
1320     !  Normally when making measurements they should be preceeded
1330     !  by a DIGITIZE.  But, because I have the scope setup like I
1340     !  want it from previous sub programs and I will be using the
1350     !  statistics, I will not use the DIGITIZE command.
1360     !
1370     PRINT "  Making measurements on Channel 1.  "
1380     PRINT "     Measuring for 5 seconds."
1390     OUTPUT @Scope;":measure:source channel1"
1400     OUTPUT @Scope;":measure:statistics on;sendvalid on"
1410     OUTPUT @Scope;":measure:pwidth;vpp"
1420     WAIT 5       ! Give the measurements a chance to build up values
1430     OUTPUT @Scope;"stop"  ! Stop; match on screen values with returned.
1440     OUTPUT @Scope;":measure:results?"
1450     ENTER @Scope USING "%,K";A(*)
1460     Eng(A(*),R(*),Ex$(*)) ! Convert to Engineering Notation
1470     CLEAR SCREEN
1480     PRINT "  The results of the positive pulse width measurement are;"
1490     PRINT
1500     Result_state(R(2),M$) ! Interpret the Result State value
1510     PRINT "The current value is ";R(1);Ex$(1)
1520     PRINT "The state value is ";R(2),"It means, ";M$
1530     PRINT "The minimum value is ";R(3);Ex$(3)
1540     PRINT "The maximum value is ";R(4);Ex$(4)
1550     PRINT "The average value is ";R(5);Ex$(5)
1560     PRINT "The standard deviation is ";R(6);Ex$(6)
1570     PRINT "The number of measurements is ";R(7);Ex$(7)
1580     PRINT "  The results of the peak to peak measurement are;"
1590     PRINT
1600     Result_state(R(9),M$) !  Interpret the Result State Value.
1610     PRINT "  The current values is ";R(8);Ex$(8)
1620     PRINT "  The state value is ";R(9),"It means, ";M$
1630     PRINT "  The minimum value is ";R(10);Ex$(10)
1640     PRINT "  The maximum value is ";R(11);Ex$(11)
1650     PRINT "  The average value is ";R(12);Ex$(12)
1660     PRINT "  The standard deviation is ";R(13);Ex$(13)
1670     PRINT "  The number of measurements is ";R(14);Ex$(14)
1680     PRINT
1690     PRINT "  Press continue when ready to continue."
1700     PAUSE
1710     CLEAR SCREEN
1720     OUTPUT @Scope;"run"
1730   SUBEND
1740   !
1750   SUB Rt_2g(@Scope)
1760      !**************************************************************!
```

```
1770     !* This sub program is just like "Rt_4g" except it samples at *!
1780     !* 2 GSa/s.                                                   *!
1790     !*************************************************************!
1800    CLEAR SCREEN
1810    PRINT "Setup Scope for 2 GSa/Sec."
1820    OUTPUT @Scope;":acquire:srate 2E+9"
1830    OUTPUT @Scope;":acquire:interpolate on"
1840    PRINT
1850    PRINT "  When ready to make the measurements press continue."
1860    PAUSE
1870    CLEAR SCREEN
1880  SUBEND
1890  !
1900  SUB Eng(N(*),A(*),Ex$(*))
1910    !********************************************************!
1920    !*  This sub program converts a # to engineering notation *!
1930    !********************************************************!
1940    S=SIZE(N,1)
1950    FOR C=1 TO S
1960      SELECT N(C)
1970      CASE >.999
1980        A(C)=N(C)
1990      CASE >9.99E-4
2000        A(C)=N(C)/1.E-3
2010        Ex$(C)="m"
2020      CASE >9.99E-7
2030        A(C)=N(C)/1.E-6
2040        Ex$(C)="u"
2050      CASE >9.99E-10
2060        A(C)=N(C)/1.E-9
2070        Ex$(C)="n"
2080      CASE >9.99E-13
2090        A(C)=N(C)/1.E-12
2100        Ex$(C)="p"
2110      CASE >9.99E-16
2120        A(C)=N(C)/1.E-15
2130        Ex$(C)="f"
2140      CASE ELSE
2150        A(C)=N(C)
2160      END SELECT
2170    NEXT C
2180  SUBEND
2190  !
2200  SUB Result_state(N,M$)
2210    !*******************************************************!
```

```
2220    !*  This sub program interprets the Result State Value   *!
2230    !*********************************************************!
2240    SELECT N
2250    CASE 0
2260      M$="Result Correct"
2270    CASE 1
2280      M$="Result Questionable"
2290    CASE 2
2300      M$="Result Less than or Equal to"
2310    CASE 3
2320      M$="Result Greater than or Equal to"
2330    CASE 4
2340      M$="Result Invalid"
2350    CASE 5
2360      M$="Edge Not Found"
2370    CASE 6
2380      M$="Max. q level not found"
2390    CASE 7
2400      M$="Min. q level not found"
2410    CASE 8
2420      M$="Requested Time not found"
2430    CASE 9
2440      M$="Voltage not found"
2450    CASE 10
2460      M$="Top and Base are equal"
2470    CASE 11
2480      M$="Measurement zone too small"
2490    CASE 12
2500      M$="Lower Threshold"
2510    CASE 13
2520      M$="Upper Threshold"
2530    CASE 14
2540      M$="Bad Upper/Lower combination"
2550    CASE 15
2560      M$="Top not on waveform"
2570    CASE 16
2580      M$="Base not on waveform"
2590    CASE 17
2600      M$="Completion Criteria not reached"
2610    CASE 18
2620      M$="Invalid Signal for measurement"
2630    CASE 19
2640      M$="Source Signal not displayed"
2650    CASE 20
2660      M$="Clipped High"
```

```
2670    CASE 21
2680      M$="Clipped Low"
2690    CASE 22
2700      M$="Clipped High and Low"
2710    CASE 23
2720      M$="All Holes"
2730    CASE 24
2740      M$="No Data on Screen"
2750    CASE 25
2760      M$="Cursor not on screen"
2770    CASE 26
2780      M$="Measurement Aborted"
2790    CASE 27
2800      M$="Measurement timed out"
2810    CASE 28
2820      M$="No measurement to track"
2830    CASE ELSE
2840    END SELECT
2850  SUBEND
```

Results? Measurement Example

```
10 !  RE-SAVE "RESU_7XX"!Operation of SENDValid & STATistics on RESULTS?
20 !
30 !*********************************************************************!
40 !********************  Main Program, Rev. 1.18  ***********************!
50 !*********************************************************************!
60    Readme
70    Initscope(@Scope)
80    True_rep(@Scope)
90    Measure(@Scope)
100   PRINT "End of Program -- Results are on your printer."
110   BEEP 15,2
120         !*************************************************************!
130   END   !*                      End of Main Program                  *!
140         !*************************************************************!
150                                                                       !
160         !*************************************************************!
170         !*                      Begin Sub Programs                   *!
180         !*************************************************************!
190   SUB Readme
200     !*****************************************************************!
210     !*This sub program writes user program information to the screen.*!
220     !*****************************************************************!
230     CLEAR SCREEN
240     PRINT "This example program will setup and measure the cal."
250     PRINT "signal from the 54721A in the ET mode.  "
260     PRINT
270     PRINT "It measures the Positive Pulse Width with Statistics."
280     PRINT "Then uses the *RESULTS?* to report over the HPIB."
290     PRINT
300     PRINT "The report from the RESULTS? varies depending on the status"
310     PRINT "of STATISTICS ON|OFF and SENDValid ON|OFF."
320     PRINT
330     PRINT "This program will print the results for each of the cases."
340     PRINT
350     PRINT "The program assumes that the system is configured such that:"
360     PRINT
370     PRINT "   HP-IB interface is at address 7."
380     PRINT "   Scope is at address 7."
390     PRINT "   A 54721A is installed into slots 1 & 2."
400     PRINT "   Printer at 701"
410     PRINT
```

```
420     PRINT "If these addresses are incorrect, break program and set addresses"
430     PRINT "as needed in the ASSIGN statements."
440     PRINT
450     PRINT "PRESS continue to run program"
460     PAUSE
470     CLEAR SCREEN
480   SUBEND
490     !
500   SUB Initscope(@Scope)
510     !****************************************************************!
520     !*  This sub program initializes the I/O and scope.             *!
530     !****************************************************************!
540     ASSIGN @Scope TO 707
550     CLEAR @Scope                      !clear HP-IB interface
560     OUTPUT @Scope;"*cls"
570     OUTPUT @Scope;"*RST"              !reset scope to default config
580     OUTPUT @Scope;":SYSTEM:HEADER OFF"!turn off header
590     CLEAR SCREEN
600   SUBEND
610   !
620   SUB True_rep(@Scope)
630     !****************************************************************!
640     !*  This sets up the scope to look at the calibration signal of *!
650     !*  the 54721A in the ET (equivalen time mode).                 *!
660     !****************************************************************!
670     PRINT "  Connect the HP 54721A Calibrator Output to the Input  "
680     PRINT "  of the 54721A.                                        "
690     PRINT
700     PRINT "  Press continue when ready to continue.  "
710     PAUSE
720     CLEAR SCREEN
730     OUTPUT @Scope;":channel1:display on"
740     !********** Turn on the  calibrator signal on the 21 plug-in.
750     OUTPUT @Scope;":channel1:output on"
760     OUTPUT @Scope;":acquire:mode etime"
770     OUTPUT @Scope;":autoscale"
780     OUTPUT @Scope;":display:persistence infinite"
790     WAIT 5
800     PRINT "  The displayed waveform on the 54720A is the true "
810     PRINT "  representation of the 54721A cal. signal."
820     PRINT
830     PRINT "  The HP 54720A is in equivelent time mode:"
840     PRINT "      Sampling rate is 500 MSa/S,"
850     PRINT "      Analog Bandwidth is 1.1 GHz"
860     PRINT
```

```
870  SUBEND
880  !
890  SUB Measure(@Scope)
900    !***************************************************************!
910    !* This sub program will make a +width measurement.           *!
920    !* It will also report the mean and standard deviations.      *!
930    !***************************************************************!
940    CLEAR SCREEN
950    PRINT "  Measuring Waveform and Reporting Results.  "
960    REAL R(1:12)
970    !***************************************************************!
980    !  Normally when making measurements, they should be preceeded  !
990    !  by a DIGITIZE.  But, because I have the scope setup like I   !
1000   !  want it from previous sub programs and I will be using the   !
1010   !  statistics, I will not use the DIGITIZE command.             !
1020   !***************************************************************!
1030   OUTPUT @Scope;":measure:source channel1"
1040   PRINTER IS 701
1050   FOR C=1 TO 4
1060     OUTPUT @Scope;"run"
1070     MAT R= (0)
1080     OUTPUT @Scope;":measure:statistics ";INT(C/3)
1090     OUTPUT @Scope;":measure:sendvalid ";C MOD 2
1100     OUTPUT @Scope;":measure:statistics?"
1110     ENTER @Scope;St$
1120     OUTPUT @Scope;":measure:sendvalid?"
1130     ENTER @Scope;Sv$
1140     OUTPUT @Scope;":measure:pwidth"
1150     OUTPUT CRT;"Measuring for 20 seconds to get good stats."
1160     WAIT 20   !* Give the measurements a chance to build up values *!
1170     OUTPUT @Scope;"stop"!* See that values match ON SCREEN & OVER HPIB *!
1180     OUTPUT @Scope;":measure:results?"
1190     ENTER @Scope USING "%,K";R(*)
1200     OUTPUT CRT;"Printing Results to your printer"
1210     PRINT
1220     PRINT "Statistics is ";St$,"SendValid is ";Sv$
1230     PRINT
1240     PRINT "First value is ";R(1)
1250     PRINT "Second value is ";R(2)
1260     PRINT "Third value is ";R(3)
1270     PRINT "Fourth value is ";R(4)
1280     PRINT "Fifth value is ";R(5)
1290     PRINT "Sixth value is ";R(6)
1300     PRINT "Seventh value is ";R(7)
1310     WAIT 5
```

```
1320        CLEAR SCREEN
1330      NEXT C
1340      PRINTER IS CRT
1350    SUBEND
```

Learn String Example Program

```
10    !RE-SAVE "LSTG7XX2"  !HP Basic for HP-IB interface, rev 2.0
20    !*****************************************************************
30    !*  This program reads and returns the learn string from and to a  *
40    !*  547XX Oscilloscope.                                            *
50    !*                    ********************                         *
60    !*                    ********************                         *
70    !*                     Begin MAIN PROGRAM                          *
80    !*****************************************************************
90    COM /Io/ @Scope,Hpib
100   Readme                        !Description of the program
110   Initscope                     !initialize interface and scope
120   Length=FNStsize               !find setup string size.
130   Get_learnstr(Length)          !save 3 configurations on disk
140   Recall_learnstr(Length)       !select & recall 1 of 3 setups
150   PRINT
160   BEEP 15,1
170   PRINT "program done"
180     !***************************************************************
190     !*                    End of Main Programs                      *
200     !***************************************************************
210   END
220     !***************************************************************
230     !*                    Begin Sub Programs                        *
240     !***************************************************************
250   SUB Initscope
260     !**********************************************************
270     !*  This sub program initializes the INTERFACE AND SCOPE  *
280     !**********************************************************
290     COM /Io/ @Scope,Hpib
300     Hpib=7
310     Scope=7
320     ASSIGN @Scope TO Hpib*100+Scope  !scope address
330     CLEAR Hpib                       !clear HPIB interface
340     OUTPUT @Scope;"*RST"             !set scope to default config
350     OUTPUT @Scope;":AUTOSCALE"       !AUTOSCALE
360     OUTPUT @Scope;":SYST:HEAD OFF"
370     OUTPUT @Scope;"*OPC?"            !wait for scope to finish auto
380     ENTER @Scope;Opc
390   SUBEND
400     !
410   SUB Readme
```

```
420     !***********************************************************************
430     !*  This sub program displays a message about the program for the      *
440     !*  user.                                                              *
450     !***********************************************************************
460     CLEAR SCREEN
470     PRINT "This sample program will prompt the user to set up the"
480     PRINT "scope in three different configurations and will store"
490     PRINT "them to the computer disk.  Any of the three configurations"
500     PRINT "may then be recalled from the disk and sent to the scope"
510     PRINT
520     PRINT "The program assumes that the system is configured such that:"
530     PRINT "      HP-IB interface is at address 7"
540     PRINT "      scope is at address 7"
550     PRINT "      a signal is attached to channel 1"
560     PRINT
570     PRINT "If these addresses are incorrect, break program and set addresses"
580     PRINT "as needed in the Initialize in the ASSIGN statements."
590     PRINT
600     PRINT "Press CONTINUE when ready to start.  Scope will first autoscale"
610     PRINT "on signal on channel 1 and will then prompt for user to setup"
620     PRINT "scope as desired before saving configurations in computer."
630     PRINT
640     PAUSE
650     CLEAR SCREEN
660   SUBEND
670   !
680   SUB Get_learnstr(Length)
690     !*************************************************************
700     !*  This sub program will get the learn string from the 547XX *
710     !*  and place it in SET$.  Then it will create a BDAT file    *
720     !*  called "JSETUPS" which holds 3 records.  If this file is  *
730     !*  already created it will be PURGED!                        *
740     !*************************************************************
750     COM /Io/ @Scope,Hpib
760     ON ERROR CALL Ertrap
770     CREATE BDAT "JSETUPS",3,Length    !create 3 files for 3 different
780                                       !setups.
790     ALLOCATE Set$[Length]             !temp variable to hold string.
800     ASSIGN @Path TO "JSETUPS"         !open file
810     FOR I=1 TO 3
820       CLEAR SCREEN
830       LOCAL @Scope
840       PRINT "PLEASE HAVE SETUP #";I;" READY AND PRESS RETURN"
850       INPUT A$
860       OUTPUT @Scope;":SYSTEM:SETUP?"   !query learnstring from scope
```

```
870       ENTER @Scope USING "-K";Set$      !read learn string from scope
880       IF Set$[1;1]="#" THEN
890         OUTPUT @Path,I;Set$             !store setup string to disk
900       ELSE
910         CLEAR SCREEN
920         PRINT "Received bad data.  No setup saved."
930       END IF
940     NEXT I
950     ASSIGN @Path TO *                                       !close file
960     DEALLOCATE Set$
970   SUBEND
980   !
990   SUB Ertrap
1000    !***********************************************************
1010    !*  The program will branch to this Error Trap if the      *
1020    !*  ON ERROR is ON and an error occurs.  It reset the      *
1030    !*  ON ERROR to OFF the return to where it was called.     *
1040    !***********************************************************
1050    IF ERRN=54 THEN !  Error 54 is Duplicate File Name
1060      PURGE "JSETUPS"
1070      OFF ERROR
1080    ELSE
1090      CLEAR SCREEN
1100      PRINT ERRM$
1110      BEEP
1120      PAUSE
1130    END IF
1140  SUBEND
1150  !
1160  SUB Recall_learnstr(Length)
1170    !
1180    !  This sub program lets the user select which of the 3 setups
1190    !  that have been stored on the disk in "JSETUPS1, 2, or 3 to
1200    !  use to setup the scope.  It will loop until the user selects
1210    !  (E) to exit.
1220    !
1230    COM /Io/ @Scope,Hpib
1240    ASSIGN @Path TO "JSETUPS"                  !open file
1250    ALLOCATE Set$[Length]                      !create temp variable.
1260    Done=0
1270    REPEAT
1280      CLEAR SCREEN
1290      PRINT "Please enter (1) to recall setup 1"
1300      PRINT "             (2) to recall setup 2"
1310      PRINT "             (3) to recall setup 3"
```

```
1320      PRINT "              (E) to exit"
1330      INPUT A$
1340      SELECT UPC$(A$)
1350      CASE "1","2","3"
1360        ENTER @Path,VAL(A$);Set$                  !read data from disk.
1370        IF Set$[1;1]="#" THEN                     !Have good data.
1380    !
1390    ! Add command header to setup string and send entire string to scope.
1400    !
1410          OUTPUT @Scope USING "#,K";":SYSTEM:SETUP ";Set$
1420        ELSE
1430          CLEAR SCREEN
1440          PRINT "Received bad data, no string entered."
1450        END IF
1460      CASE "E"
1470        Done=1
1480      END SELECT
1490    UNTIL Done
1500    DEALLOCATE Set$
1510    ASSIGN @Path TO *
1520  SUBEND
1530  !
1540  DEF FNStsize
1550      !
1560      !The setup string size can varry dependant upon operating system
1570      !revision.  Must read the header to determine the proper lengths.
1580      !The format of the data is #NX...X<setup data string>.  Then I
1590      !add 5 for the bdat file management headers.
1600      !
1610    COM /Io/ @Scope,Hpib
1620    DIM Psign$[1]
1630    INTEGER Length,Cnt,L
1640    ON TIMEOUT Hpib,3 CALL Tout
1650            !Set the bus timeout so if there is no/bad data, can't find
1660            !the # sign, we will stop and let the operator know.
1670    OUTPUT @Scope;":SYSTEM:SETUP?"
1680            !Query scope for the setup string.
1690    Cnt=0
1700    REPEAT
1710      ENTER @Scope USING "#,A";Psign$
1720            !Enter a character at a time until find the # sign.  It
1730            !indicates the beginning of the block header.
1740      Cnt=Cnt+1
1750            !FN must keep track of the number of characters before the #
1760            !sign for cases where the system headers are ON.
```

```
1770   UNTIL Psign$="#"
1780   ENTER @Scope USING "#,A";Psign$
1790           !Next character tells the number of digits in the header.
1800   ENTER @Scope USING "#,"&Psign$&"D";Length
1810           !Length is the number of data values to follow before the NL.
1820   ALLOCATE Temp$[Length+1]
1830   ENTER @Scope USING "#,-K";Temp$
1840   L=7+Cnt+VAL(Psign$)+Length
1850   DEALLOCATE Temp$
1860   RETURN L
1870 FNEND
1880 !
1890 SUB Tout
1900     !Branching here says that the HPIB bus was idle for 3 seconds.
1910     !This would be cause by reaching the end of the setup data without
1920     !finding a # sign.
1930     CLEAR SCREEN
1940     PRINT "Bad Data, query aborted."
1950     BEEP
1960     PAUSE
1970 SUBEND
```

Service Request Example Program

```
10    !  RE-SAVE "SRQ_7XX"
20    !  This program sets the Event Status Enable Register and the
30    !  Service Request Enable Register so that an error will cause
40    !  a service request.  It also saves a setup to a setup memory
50    !  and recalls that setup.
60    DIM Query$[15],Command$[15],Q$[9000]
70    COM /Err/ Hpib,Scope
80    COM /S/ @S
90    Hpib=7
100   CLEAR Hpib
110   Scope=7
120   Saddr=Hpib*100+Scope        !Sets the I/O address.
130   Readme
140   ASSIGN @S TO Saddr
150   ON INTR Hpib,15 CALL Ermsg  !Tells computer where to go on an error
160   CLEAR Saddr
170   OUTPUT @S;"*ESE 60;*SRE 32;*CLS"
180   !   *ESE XX sets the Event Status Enable Regiser.
190   !         32 => CME or Command Error
200   !         16 => EXE or Execution Error
210   !          8 => DDE or Device Dependent Error
220   !          4 => QYE or Query Error
230   !         -----------------------------------
240   !         60
250   !   *SRE XX sets the Service Request Enable Register.
260   !         32 => ESB or Event Status Bit
270   ENABLE INTR Hpib;2          !Allows the HPIB to interupt the computer.
280   Saveset$="*SAV "            !This command is missing the parameter, 1.
290   Recallset$="*RCL 1"         !This recalls setup #1.
300   !
310   OUTPUT @S;Saveset$     !Send the command, causes CME until the 1 is added.
320   WAIT 1
330   LOCAL Saddr
340   Readme1
350   OUTPUT @S;Recallset$        !Recalls setup #1.
360   LOCAL Saddr
370   Readme2
380   END
390   !
400   SUB Ermsg                   !Error Trap
410     COM /S/ @S
```

```
420    COM /Err/ Hpib,Scope
430    DIM E$[50]
440    PRINT "An error occured.",
450    Sp=SPOLL(707)
460    IF BIT(Sp,6) THEN
470    ! Testing bit 6 will tell us if the scope is the source of the interupt.
480      OUTPUT @S;"*ESR?"!Reads then clears the Standard Event Status Register
490      ENTER @S;J
500      Srq_type(J)          !Call to the SRQ interpreter subprogram.
510      Done=0
520      REPEAT
530    !  Read the Error Queue to determine the specific error.  Repeat reading
540    !  until the Queue is empty.  Each time the queue is read the error will
550    !  be reported or the message 'THERE ARE NO MORE ERRORS' will be returned.
560        OUTPUT @S;":SYSTEM:ERROR? STRING"
570        ENTER @S;E$
580        IF E$[1;1]="0" THEN
590          PRINT "THERE ARE NO MORE ERRORS."
600          OUTPUT @S;"*CLS"    !Clears all event registers and queues except
610                              !the output queue.
620          ENABLE INTR Hpib;2
630          Done=1
640          Readme3
650        ELSE
660          PRINT E$
670        END IF
680      UNTIL Done
690    ELSE
700      CLEAR SCREEN
710      PRINT "An interupt on the HPIB has occured but it is not the "
720      PRINT "scope.  Please clear the other source of interupt before"
730      PRINT "restarting this program."
740      PAUSE
750    END IF
760  SUBEND
770  !
780  SUB Readme
790    PRINT "  This program sets the Event Status Enable Register and the"
800    PRINT "  Service Request Enable Register so that an error will cause"
810    PRINT "  a service request. "
820    PRINT
830    PRINT "  The second function of this program is to save and then recall"
840    PRINT "  the current scope setup to and from setup memory 1.  However,"
850    PRINT "  there is a bug in this program.  The save command is missing a"
860    PRINT "  parameter.  It needs a 1 after the space or '*SAV 1'."
```

```
870     PRINT
880     PRINT "  After you have seen how the SRQ's work you may edit line 280"
890     PRINT "  to save the setup."
900     PRINT
910     PRINT "  Once you have fixed the bug the program will run and save the"
920     PRINT "  current setup to setup memory 1 then pause and allow you to "
930     PRINT "  change the setup.  When you continue, the original setup will"
940     PRINT "  be restored."
950     PRINT
960     PRINT "  The expected configuration is;"
970     PRINT "      The scope is at address 7"
980     PRINT "      The HPIB is at address 7"
990     PRINT
1000    PRINT "  If the configuration is different break program and set "
1010    PRINT "  the addresses as required using the variables Hpib and "
1020    PRINT "  Scope.  Then run again."
1030    PRINT
1040    PRINT "  Press Continue when ready to resumen operation."
1050    PRINT
1060    PAUSE
1070    CLEAR SCREEN
1080  SUBEND
1090  !
1100  SUB Readme1
1110    PRINT
1120    PRINT "   The current setup has been saved in setup memory #1."
1130    PRINT
1140    PRINT "   Change the scopes setup from the front panel.  When you"
1150    PRINT "   press continue the original setup will be restored."
1160    PRINT
1170    BEEP
1180    PAUSE
1190    CLEAR SCREEN
1200  SUBEND
1210  !
1220  SUB Readme2
1230    PRINT "   The program has ended.  Thanks for trying our Save "
1240    PRINT "   and Recall setup memories."
1250    PRINT
1260    PRINT "   Now would you have the opportunity to edit the program"
1270    PRINT "   to generate and error and see how the interupt masking"
1280    PRINT "   works."
1290    PRINT
1300    PRINT "           GOODBYE.     "
1310  SUBEND
```

```
1320  !
1330  SUB Readme3
1340    PRINT
1350    PRINT "  Break the program at this time and determine the cause of "
1360    PRINT "  the error."
1370    PRINT
1380    PRINT "  Once you have fixed the cause of the error described by"
1390    PRINT "  the error code and message, rerun the program by pressing"
1400    PRINT "  RUN."
1410    PAUSE
1420    CLEAR SCREEN
1430  SUBEND
1440  !
1450  SUB Srq_type(J)
1460  !  The scope has interupted the computer and we have read the
1470  !  Standard Event Status Register.  Now the value, J, that was
1480  !  read by the *ESR? will be evaluated to determine why the
1490  !  SRQ was generated.
1500    PRINT "ESR value is ";J
1510    SELECT J
1520    CASE 32
1530      PRINT "32 => CME or Command Error."
1540    CASE 16
1550      PRINT "16 => EXE or Execution Error."
1560    CASE 8
1570      PRINT "8 => DDE or Device Dependent Error."
1580    CASE 4
1590      PRINT "4 => QYE or Query Error."
1600    END SELECT
1610  SUBEND
```

Configuration Example Program

```
10    !RE-SAVE "CONFIG"   !This is an RMB and IBasic Program.
20    !
30    !It queries the scope to determine the configuration and then
40    !prints it to the crt.  It assumes that the scope is at HPIB
50    !
60    DIM Mframe$[13],Slot$(1:4)[13]
70    OUTPUT 707;":SYSTEM:HEADER ON"
80    OUTPUT 707;":SYSTEM:LONGFORM ON"
90    !
100   !***** DETERMINE THE FRAME MODEL NUMBER ******!
110   !
120   OUTPUT 707;":MODEL? FRAME"
130   ENTER 707;Mframe$
140   !
150   !*****  DETERMINE THE PLUG-INS AND THEIR LOCATIONS ********!
160   !
170   FOR I=1 TO 4
180     OUTPUT 707 USING "K";":MODEL? PLUGIN";I
190     ENTER 707;Slot$(I)
200   NEXT I
210   !
220   !*****  REPORT THE MAINFRAME MODEL # AND PLUG-INS *******!
230   !
240   CLEAR SCREEN
250   PRINT "The Main frame is ";Mframe$
260   PRINT
270   PRINT "The plug-in in slot 1 is ";Slot$(1)
280   PRINT "The plug-in in slot 2 is ";Slot$(2)
290   PRINT "The plug-in in slot 3 is ";Slot$(3)
300   PRINT "The plug-in in slot 4 is ";Slot$(4)
310   PRINT
320   PRINT "End of Program"
330   END
```

Limit Test Example Program

```
10    ! MLIM.ibw
20    !
30    !       Copyright: (c) 1993, Hewlett-Packard Co.  All rights reserved.
40    !     Contributor: Colorado Springs Division
50    !         Product: Throughput Application
60    !
70    ! $Revision$        3.0
80    ! $Date$            6-14-93
90    ! $Author$          Ed Mierzejewski
100   !
110   ! Structure Chart: None
120   !     Description: This Uses Measure Limit Testing to make 3 measurements
130   !                  on 10 successive pulses at a 10 Hz rate.
170   !
180   !  Considerations: None
200   !    Main routine: Begin_main
210   !    Sub-routines: None
220   !       Functions: None
230   !    Sub-programs: Readme, Set_paths, Set_scope, Meas, Tcount
240   !
250 Variable_list:!
280 Begin_main:   !
290   CALL Readme
300   CALL Set_paths(@Scope,Isc)
310   CALL Set_scope(@Scope)
320   CALL Meas(@Scope,Isc)
330 End_of_main:  !
340   END
350   !
360 Begin_subs:  !
370   !
380   SUB Readme
390   !
400   ! Description: Readme writes instructions and information at the
410   !              beginning of the program for the user to ensure
420   !              proper setup prior to continuing the program.
430   !
440   !  Parameters: None
450   !
460     CLEAR SCREEN
461     PRINT TABXY(5,5)
```

```
470     PRINT "MLIM.ibw uses a HP8131 in the burst mode to manualy start a "
471     PRINT "burst of 10, 2ns pulses with a 99.9 ms period, 1 V amp."
472     PRINT
473     PRINT "    It will work with any similar signal, including the "
474     PRINT "    front panel cal."
475     PRINT
480     PRINT "It makes a Vpp, Risetime, and Positive pulse width measurement"
490     PRINT "on each and reports the mean after all 10 measurement sets are"
491     PRINT "complete."
500     PRINT
510     PRINT "There are NO specific plug-ins required, except a suitable 1"
520     PRINT "must be in channel 1.  (Program was developed using a '13A"
521     PRINT "installed in slot 1 of a 54710A."
530     PRINT
540     PRINT "The HPIB card is assumed to be at interface select code 7 and"
550     PRINT "the scope is at address 7."
560     PRINT
570     PRINT "Ensure all of this is correct before continuing."
571     PRINT
580     PRINT "press continue when through reading this."
590     PAUSE
600     CLEAR SCREEN
610   SUBEND
620   !
960   SUB Set_scope(@S)
970   !
980   ! Description: Set_scope has 2 parts:
990   !               1 -- initialize the scope and i/o.
1010  !               2 -- set for RT acqires and measurement of the pulses.
1020  !
1030  !  Parameters:
1040  !      Passed: (@S) @Scope = specific scope's address,
1050  !    Internal:
1060  !
1070  ! Modified Variables: None
1080  !
1090 Part_1:                               ! Initialize for RT to measure pulses.
1100    OUTPUT @S;"*rst;*cls"
1101    OUTPUT @S;":opee 256"               ! Unmasks the Lim.Tst. Comp. bit.
1104    OUTPUT @S;"*sre 128"                ! Unmasks the oper bit, see Ltest.
1105    OUTPUT @S;":disp:grat fram"
1110    OUTPUT @S;":blan chan2;view chan1"  ! 54710 only has 2 chan's avail.
1120    OUTPUT @S;":acq:mode rtim;srat 2E9;poin 512"
1130    OUTPUT @S;":chan1:bwl off;disp on;inp dc50;offs 0;prob 1,rat;scal .5"
1140    OUTPUT @S;":tim:pos 10E-9;scal 2.5E-9"
```

```
1150    OUTPUT @S;":trig:swe trig;sour trig1;lev trig1,.5"
1190 Part_2:                                          ! Set Measurements
1200    OUTPUT @S;":meas:send off;stat on;sour chan1"
1201    !
1202    ! Turn off sendvalid, on statistics, and sets source to channel 1
1203    !
1210    OUTPUT @S;":meas:vpp;ris;pwid"
1220  SUBEND
1230  !
1240  SUB Set_paths(@Scope,Isc)
1250  !
1260  ! Description: Set_paths simply assigns HPIB select code to be 7 and
1261  !              the scope address to be 7.
1270  !
1280  !  Parameters:
1310  !    Passed:   @Scope = I/O path 707
1320  !              scope = the HPIB address that the scope is selected to.
1330  !  Internal:   Isc   = the interface select code for the HPIB card.
1350  ! Modified Variables: @Scope
1360  !
1400    CLEAR SCREEN
1420    Isc=7
1430    Scope=7
1580    ASSIGN @Scope TO Isc*100+Scope
1610  SUBEND
1620  !
1630  SUB Meas(@S,Isc)
1640  !
1641  ! Description: The scope is setup and waiting to make continuous meas's.
1642  !              1 -- Setup On interupt so Lim. Tst. Comp. gives SRQ.
1643  !              2 -- Setup Limit test.
1644  !              3 -- Set RUN Limit Tests.
1645  !              4 -- Report results.
1646  !
1647  !  Parameters:
1648  !      Passed: (@S) @Scope = specific scope's address,
1650  !              COM /For_cnt/ INTEGER num_acq, M
1651  !
1653  !    Internal: Results(*) = array of values returned from a RESULTS?
1654  !                           Value 4 of each set is the mean.  Therefore,
1655  !                           Results(4), (13), and (22) are the ones of
1656  !                           interest.
1657  !              M = measurement sets requested.
1658  !              Num_acq = the termination variable.
1661  !
```

```
1662  ! Modified Variables: Num_acq
1663  !
1664  ! Calls sub programs: Tcount
1665  !
1871    COM /For_cnt/ INTEGER Num_acq,M,S  ! need to pass num_acq on intr.
1872    REAL Results(1:27)                 ! 9 parameters per measurement.
1900    M=10
1901    Num_acq=0
1910    CLEAR SCREEN
1920 Part1:                                ! Setup interupt
1930    ON INTR Isc,9 CALL Tcount
2000 Part2: !
2001    OUTPUT @S;":ltes:sour 1;fail nev;mnf pass;run wav,";M
2002    OUTPUT @S;":ltes:sour 2;fail nev;mnf pass;run wav,";M
2003    OUTPUT @S;":ltes:sour 3;fail nev;mnf pass;run wav,";M
2010    OUTPUT @S;":stop;cdis"
2020    OUTPUT @S;":ltes:test on"
2060 Part3: !
2068    ENABLE INTR Isc;2
2069    OUTPUT @S;":run"                   ! scope will wait for triggers.
2070    PRINT "If using the 8131A, Start Generator NOW."
2071    REPEAT                             ! wait for limit test complet.
2076    UNTIL Num_acq=M
2077 Part4: !
2080    OUTPUT @S;":meas:res?"             ! read summary of measurements.
2081    ENTER @S;Results(*)
2082    CLEAR SCREEN
2083    PRINT "        The results are;"
2085    PRINT "the vpp mean is ";Results(4);","
2086    PRINT "the rise time mean is ";Results(13);", and"
2087    PRINT "the +width mean is ";Results(22);"."
2088    PRINT
2089    PRINT Results(*),
2091  SUBEND
2100  !
2110  SUB Tcount
2111  !
2112  ! Description: Tcount will set Num_acq to the stop value when the Lim.
2113  !              Test Complete interupt occurs.
2118  !  Parameters:
2119  !      Passed:
2120  !              COM /For_cnt/ INTEGER Num_acq,M
2121  !              Num_acq = the variable used to terminate at proper number.
2122  !              M = the number of acquisitions wanted, termination value.
2124  !    Internal: None
```

```
2129  !
2130  ! Modified Variables: num_acq
2131  !
2132  ! Calls sub programs: None
2133  !
2134    COM /For_cnt/ INTEGER Num_acq,M,S
2135    PRINT "hello"
2137    Num_acq=M
2138  SUBEND
```

Index